Information Management – Extensible Access Method (XAM) – Part 2: C API

Version 1.01

TECHNICAL POSITION

June 19, 2009

Revision History

Version	Date	Originator	Sections	Comments
1.0	7/9/08	M. McMinn	All	Released and approved by SNIA membership on June 23; changed to *Technical Position* document.
1.01	6/19/09	M. McMinn	All	Incorporated errata from SNIA XAM v1 CSpec v8.doc; added Acknowledgements to Foreword.

Contents

Figures

Tables

Foreword

Parts of this Standard

This standard is subdivided in the following parts:

- Information Management – Extensible Access Method (XAM) – Part 1: Architecture
- Information Management – Extensible Access Method (XAM) – Part 2: C API
- Information Management – Extensible Access Method (XAM) – Part 3: Java API

SNIA Web Site

Current SNIA practice is to make updates and other information available through their web site at http://www.snia.org

SNIA Address

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the Storage Networking Industry Association, 500 Sansome Street, Suite #504, San Francisco, CA 94111, U.S.A.

Acknowledgements

The SNIA FCAS (Fixed Content Aware Storage) Technical Working Group, which developed this standard, would like to recognize the significant contributions made by the following members:

Alan Yoder, Aloke Guha, Avishai Hochberg, Ben Isherwood, Cristian Teodorescu, David Black, David Slik, David Sobeck, Drew McDaniel, Jered Floyd, James Pinkerton, Jim Carlson, Kalman Meth, Kristina Tripp, Lance Evans, Leeat Ramati, Mark Carlson, Michael Allison, Michael Kilian, Mike Horgan, Nick Maliwacki, Paul Monday, Peter Cudhea, Rich Ramos, Sacha Arnoud, Scott Ostapovicz, Steve Quinn, Steve Vernon, Toby Marek, Wayne Hineman, and Zoran Cakeljic.

Introduction

Purpose and Audience

This document forms part of the XAM Software Development Kit (SDK). It is a complete reference document for C application development using the XAM API. It is intended for experienced programmers, for those developing applications that interface with storage systems that support the XAM API, and for those developing components of the XAM Library itself.

For an overview of the SNIA XAM, refer to the Business Overview chapter in the [XAM-ARCH].

Organization

The chapter contents of this document are described as follows:

Chapter	Contents
Chapter 1, “Scope”	Defines the subject of the document and the aspects covered.
Chapter 2, “Normative References”	Lists the referenced documents that are indispensable for the application of this document.
Chapter 3, “Terms and Conventions”	Defines the terms and conventions used in this document.
Chapter 4, “C API Overview”	Contains an overview of the C API.
Chapter 5, “Public C API Reference”	Contains a reference guide to the public C API for applications.
Chapter 6, “Private (VIM) C API Reference”	Contains a reference guide to the private C API for the VIMs.
Annex A, “(normative) Public Header Files”	Contains the header files for the public C API.
Annex B, “(normative) Private (VIM) Header Files”	Contains the header files for the private (VIM) C API.
Annex C, “(normative) C API Toolkit”	Describes toolkit methods to simplify some common operations within the C API.
Annex D, “(informative) C API Method Mapping”	Lists the methods in [XAM-ARCH] and the corresponding method names for the C binding.

1 Scope

This part of the XAM standard specifies the syntax of the C application programming interface (C API). It applies to programmers who are generating XAM applications in the C programming language. It also applies to storage system vendors who are creating vendor interface modules (VIMs) in the C programming language.

This document does not normatively specify the semantics of the interfaces; the specification of the semantics in the XAM standard is contained in the XAM Architecture Specification [XAM-ARCH]. Any semantics described in this document are intended to be informative and to simplify the understanding of the interfaces described herein.

2 Normative References

The following referenced document is indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[XAM-ARCH] “Information Management - Extensible Access Method (XAM) - Part 1: Architecture”, SNIA draft specification.

3 Terms and Conventions

3.1 Terms

For the purposes of this document, the definitions in the [XAM-ARCH] apply.

3.2 Conventions

Conventions used in this document include the following:

Convention	Description
Note:	Contains additional or useful informative text.
CAUTION:	Indicates that you should pay careful attention to the probable action, so that you may avoid system failure or harm.
`Fixed-width text`	Indicates text that you enter at a keyboard or text that is displayed on an output device, such as a screen. This convention is most commonly used for command syntax and examples.
Italicized text	Indicates a property or field name, i.e., *.xset.xuid*.

4 C API Overview

4.1 Basic XAM concepts

As an interface, XAM abstracts access methods from storage and provides a globally flat namespace. This interface supports the mobility of information, independent from storage, to allow longevity, distribution, and management of information. The XAM interface is intended to achieve interoperability, storage transparency, and automation for Information Lifecycle Management-based practices, long-term records retention, and information assurance (security).

The primary design goals behind the XAM interface are as follows:

- Provide a generic interface for applications: XAM interface methods have the same syntax and semantics without regard to the underlying storage. No methods were created that “lock-in” an application to a specific storage system; in fact, the systems themselves should be semantically indistinguishable when viewed from the XAM API.
- Minimal yet complete: there was a desire to keep the interface as simple and small (e.g., have as few API methods as possible, and keep these methods easy to use and understand), yet at the same time, make sure that the methods make all forms of data manipulation possible. If functionality could have been achieved by composing other methods (in a way that sufficiently ensures performance and scalabilty), then a new method was not created for that function.
- Expose no implementation detail: the interface does not expose any internal functionality that would serve to place restrictions on storage system vendors.

XAM will consist of a set of shared libraries. The ‘topmost’ library will contain the public XAM interfaces; thus, only the topmost library will be linked to applications that wish to integrate with the XAM API. However, extension libraries may also be provided which implement higher levels of functionality (e.g., placing an export method, an import method, and a delete method in series to create a ‘move’ function). When such libraries are provided, applications may wish to link to these libraries as well.

The actual implementation of the interfaces will be in the VIMs (Vendor Interface Modules). A XAM Library may utilize one or more VIMs. The implementation details of the VIMs themselves are beyond the scope of this document. The XAM API programmer should view the VIM as an internal implementation detail and avoid coding with specific VIMs in mind, if portable code is the goal. For more detailed information on the architecture of XAM, please see [XAM-ARCH].

The architecture of the XAM SDK is briefly illustrated in Figure 1, “XAM architecture”:

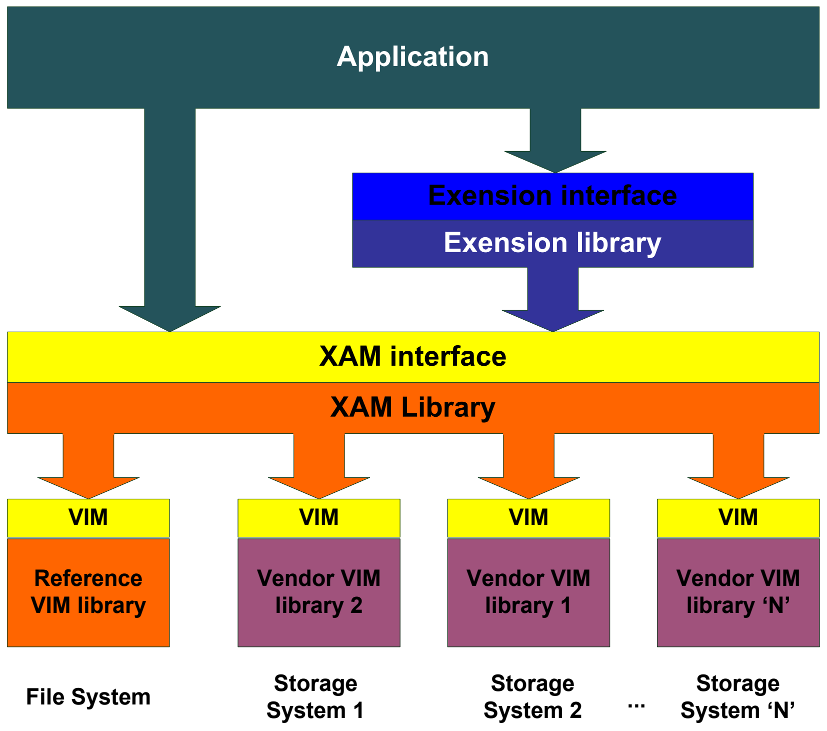

Figure 1 – XAM architecture

4.2 The XAM programming model

The XAM interface programming model supports a hierarchy of class constructs in a containment/ aggregation organization. At the top level, there is the singleton XAM object itself. Below (inside) the XAM object is one or more XSystems. Finally, each XSystem can contain XSets. Note that all of these object classes contain fields, and these fields are accessed in the same way without regard to the class of object that contains the field.

4.2.1 The XAM Library object

Pronunciation zam: The XAM Library object is the top level class for the XAM API library.

- It contains methods to get fields describing the configuration of the XAM system.
- It contains methods to set fields that controlling the configuration of the XAM system.
- It acts as a factory for XSystem instances.

4.2.2 An XSystem

Pronunciation ‘ek-sis-tm: An XSystem is the class that abstracts the connection between the application and storage system, and is a container of XSets.

- It encapsulates any resource management associated with the connection.
- It contains those methods used to authenticate operations.

- It acts as a virtual storage system, partitioning content.

In this document, we will refer to an XSystem as a single storage unit. Applications can only perform XSystem functions when an XSystem is open; otherwise, run-time errors will be generated.

4.2.3 An XSet

Pronunciation ‘ek-set: An XSet is the class that contains an application’s data and metadata.

- The XSet is assigned a globally unique identifier when stored. This globally unique identifier is called a XUID (pronounced ‘zoo-id), which stands for **X**Set **U**nique **Id**entifier.
- Data and metadata (content) stored in the XSet as fields designated as binding or nonbinding. A contract exists between the binding content of the XSet and XUID, such that if any binding fields in the XSet changes, a new XSet will be created with a new XUID upon successful commit. Nonbinding fields can be changed without generating a new XSet and thus has no effect on the XUID.

4.2.4 Fields (properties and XStreams)

Pronunciation feeld: A field is the construct where XSets, XSystems, and XAM objects store actual data and metadata. Fields have a number of attributes, which are listed below:

- Fields have names: Field names are assigned by the creator of the field.
- Fields have types: Field types are assigned by the creator of the field.
- Fields have values: These values can be changed, but the semantics of what happens to an XSet that contains a field depends on the binding nature of the field.
- Fields have lengths: These lengths are derived from the type and value assigned to the field but cannot be directly set by the application.
- Fields can be binding or nonbinding: This attribute is assigned by the application. Note that only fields on XSets can be marked as binding.
- Fields can be read/write or readonly: These attributes are controlled by XAM and cannot be set by the application.

4.2.4.1 Type and length attributes – properties vs. XStreams

Field types are identified using MIME types. XAM defines some primitive or “simple” MIME types (stypes). These types are xam_boolean, xam_int, xam_double, xam_string, xam_datetime, and XUID. The associated MIME types are, respectively; “application/vnd.snia.xam.boolean”, “application/vnd.snia.xam.int”, “application/vnd.snia.xam.double”, “application/vnd.snia.xam.string”, “application/vnd.snia.xam.datetime”, and application/vnd.snia.xam.xuid”. These types all have fixed sizes (even the string type). Fields that have one of these MIME types are referred to as properties. Note that when setting the value of a property, the XAM API will validate that the value is of the correct type (e.g., for XUID property fields, that the value actually contains a properly formatted XUID). The mapping between field type and field length is described in Table 1, “Field stypes (a.k.a. simple types)”:

Table 1 - Field stypes (a.k.a. simple types)

stype	MIME Type	Length (in bytes)
xam_boolean	application/vnd.snia.xam.boolean	1
xam_int	application/vnd.snia.xam.int	8
xam_double	application/vnd.snia.xam.double	8
XUID	application/vnd.snia.xam.xuid	9 to 80
xam_string	application/vnd.snia.xam.string	0 to MAX_XAM_STRING
xam_datetime	application/vnd.snia.xam.datetime	0 to MAX_XAM_STRING

Other MIME types are also legal. In fact, any MIME type shall be acceptable. Fields with other MIME types (e.g., non-stypes) are referred to as XStreams. For XStream fields, the associated length shall be the number of bytes in the value. Unlike properties, XStreams are not validated. The application programmer is expected to validate that the specified value is of the specified MIME type.

4.2.4.2 Binding attribute vs. readonly attribute

Finally, we have the attributes binding and readonly. While these may seem related, they are, in fact, significantly different. To use the XAM API, one must understand the differences between these two field attributes.

Binding fields are those fields whose values participate in the contract of the XSet, binding the name of the XSet to the data of the XSet. Thus, if a field whose binding attribute is set to TRUE is changed, a new XSet will be created when storing (committing) the XSet changes and a new XUID will be generated. The original XSet (and its requisite XUID) are unchanged. Fields whose binding attribute is set to FALSE (nonbinding) can be changed without affecting the XSet/XUID contract. Thus, if an XSet only has nonbinding fields changed, the XUID is unchanged when the modified XSet is committed. Because only XSets (not XSystems or XAM objects) can be committed, this field attribute can only be set on XSet fields. The binding attribute can be set by applications.

The readonly attribute controls if the application is allowed to edit a field at all. A field with the readonly attribute set to TRUE shall generate a run-time error when any method is used to edit the field. The readonly attribute is set by XAM (by the XAM Library or the XSystem instance); applications cannot alter the readonly attribute. Note that while having a field’s readonly attribute set to TRUE may seem similar to setting the field’s binding attribute to TRUE, it is not. A field may be binding and readonly, in which case, an error will occur when trying to edit the field. A field may be binding and read/write (e.g., readonly = FALSE), in which case, the edit is allowed, but on commit of the XSet changes, a new XSet with a new XUID is created, and the original XSet/XUID pair is unchanged.

4.2.5 The XIterator

Pronunciation ek-’zi-ter-a-ter: The XIterator is a field discovery class. This interface was created because XSets, XSystems, and XAM objects can all have an arbitrary number of fields (the maximum number of fields on an XSet is 2^63-1. While not an actually arbitrary number, it is still a lot). The XIterator:

- Allows the discovery of all fields on the XSet, XSystem, or XAM object.
- Takes a prefix that allows only a subset of fields to be discovered.

4.2.6 The XAsync

Pronunciation: eks-’A-sink: The XAsync is an object used to access information about an asynchronous operation. These asynchronous operations allow applications to connect to XSystems and to read and write XSets that are associated with the XSystem without blocking, or losing control of, the thread that invokes the method. This object is returned when an asynchronous method is called, which allows applications to poll the status of the operations. The object is also passed as a parameter to any callbacks associated with an asynchronous method.

4.2.7 XAM status

Pronunciation zam ‘sta-tus: XAM status is used by all methods to indicate success or failure of the method.

4.2.8 The method hierarchy

The XAM, XSystem, and XSet classes are hierarchical in nature. An application uses a XAM method to create an XSystem instance and an XSystem instance to create an XSet instance. Different methods are available when working at each level of the hierarchy. This hierarchical relationship between the methods of the XAM API is illustrated in Figure 2, “XAM API method hierarchy”:

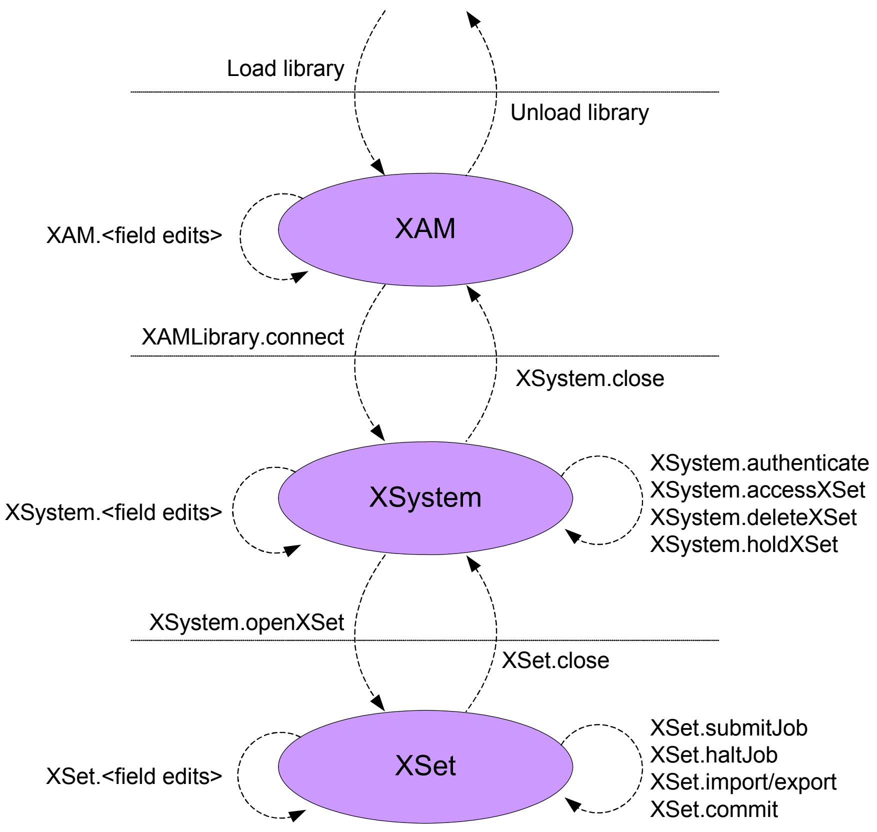

Figure 2 – XAM API method hierarchy

As illustrated in the hierarchy in Figure 2, field sets and gets can be done at any level of the hierarchy and on any XSet, XSystem, or XAM object. Property fields can be accessed directly. However, XStream fields require the use of an XStream class to read and write to the field value. The XStream supports POSIX-like semantics, and XStreams open for reading allow seeking within the XStream. In addition, the ability to enumerate the field names of all fields on the XSet, XSystem, or XAM object is also needed at all levels of the hierarchy.

Figure 3 illustrates the relationship between these field methods:

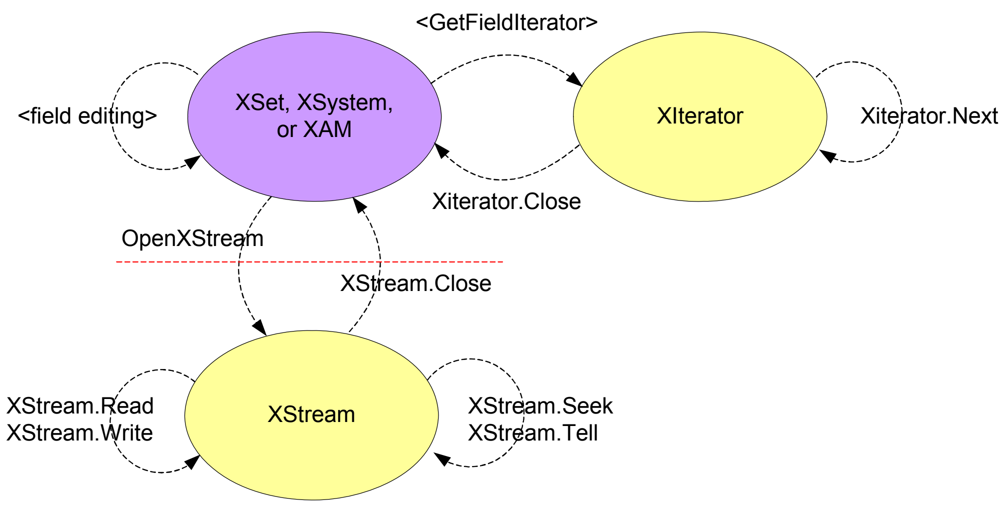

Figure 3 – XAM API field methods (includes properties and XStreams)

4.2.9 Using the XAM API – abstract samples

These are abstract examples of the types of operations that an application can perform using the XAM API. Note that these operations can be performed with either the synchronous or asynchronous methods; these methods are semantically equivalent.

4.2.9.1 Create an XSet

To write an XSet, the application must first connect to an XSystem. It then creates an XSet instance with whatever fields it wishes to add to the XSet instance. The application stores (commits) the XSet instance to the XSystem and gets an identifier (the XUID). The application then releases the resources associated with the XSet and the XSystem instances.

4.2.9.2 Read an XSet

To read an XSet, the application must first connect to an XSystem that contains the XSet. It should open the XSet using the XUID returned when the XSet was originally committed (note that this need not be the same XSystem on which the XSet was originally stored, but the XSet should reside on the XSystem to avoid a run-time error). The process of opening an XSet will generate an XSet instance. The application should read the fields from the XSet instance. The application then releases the resources associated with the XSet and the XSystem instances.

4.2.9.3 Query an XSet

To query, the application must first connect to an XSystem. It then creates an XSet instance with the specific fields needed to run the query job. A method is called on the XSet instance to start the job on the XSystem (submitJob). As the query runs, the results will be put into an XStream field on the XSet in the form of a list of XUIDs, where the application will extract the query results. To access the values of the fields, the application should read the XSets in the results as outlined in Section 4.2.9.2 (there is no need to open and close the XSystem each time an XSet is read). When completed, the application then releases the resources associated with the XSet and the XSystem instances.

5 Public C API Reference

This chapter describes the public interfaces of the XAM Library. These interfaces are intended to be used by application programmers.

5.1 Design goals

Some simple design goals were kept in mind while defining the XAM API. These goals are for all methods to:

- Return status: The use of thread local storage or thread keys for retrieving error/status information (in the XAM libraries) should not be needed.
- Have output returned by reference
- Emulate an object model
- Be thread safe
- Support asynchronous operations for operations in the data path
- Be kept to a minimum number.
- Favor compilation errors over run-time errors

5.2 Supporting data types

5.2.1 stypes

All XAM fields have type information that is described using MIME types. Complex fields require that the value of the field (the data associated with the field) be stored in an XStream. However, some predefined MIME types have also been defined for XAM fields. These MIME types (also known as simple MIME types or stypes) have data types associated with them, which allows the values to be checked at compile time.

The stypes and the data types are defined in the public header file xam_types.h and are also described below:

- **“application/vnd.snia.xam.boolean”:** This MIME type is associated with a standard boolean type, xam_boolean. A xam field with this type will have a length of 1. A valid field of this type will contain a zero (0) when FALSE or a non-zero value when TRUE.
- **“application/vnd.snia.xam.int”:** This MIME type is associated with a 64-bit integer value on all platforms, xam_int. Note that this is not the same as a standard long type. The value stored in this field can be positive or negative. A xam field with this type will have a length of 8.
- **“application/vnd.snia.xam.double”:** This MIME type is associated with a standard double precision float, xam_double. A xam field with this type will have a length of 8.
- **“application/vnd.snia.xam.xuid”:** This MIME type is associated with an 80-element byte array, xam_xuid. A valid field of this type will have a value that is a canonical XUID. A xam field with this type will have a length of 80.
- **“application/vnd.snia.xam.string”:** This MIME type is associated with a MAX_XAM_STRING element byte array, xam_string. A valid field of this type will have MAX_XAM_STRING or fewer bytes which describe the string. The encoding of a string type is UTF-8. Note that xam_strings may not contain NULLs; thus NULL termination will be used in the C API to mark the end of a

string. A xam field with this type will have a length which matches the number of bytes that describes the actual string; the terminating NULL (or other trailing bytes following the NULL) are not included in the length.

- **"application/vnd.snia.xam.datetime":** This field is associated with a MAX_XAM_STRING element byte array, **xam_datetime**. It is a ISO 8601-compliant timestamp string, UTF-8 encoded, with 4 digit years, negative years allowed, no truncated years, no week dates, no ordinal dates, no 24:00 representation of midnight, time zone designators allowed, no duration or interval formats, and a millisecond resolution.

5.2.2 XAM status type

Every method in the C API will return status. This status information will be contained in a status type. A XAM status type is a 32-bit integer, as defined below:

```
typedef int xam_status;
```

The top bit is used as a flag, while the remaining 31 bits are used to hold the status payload. The topmost bit (bit 0) is set to zero when the payload contains a value defined in this standard (standard value), and 1 when the payload contains a non-standard (vendor-specific) value. The status format is illustrated in Figure 4, "XAM status type diagram":

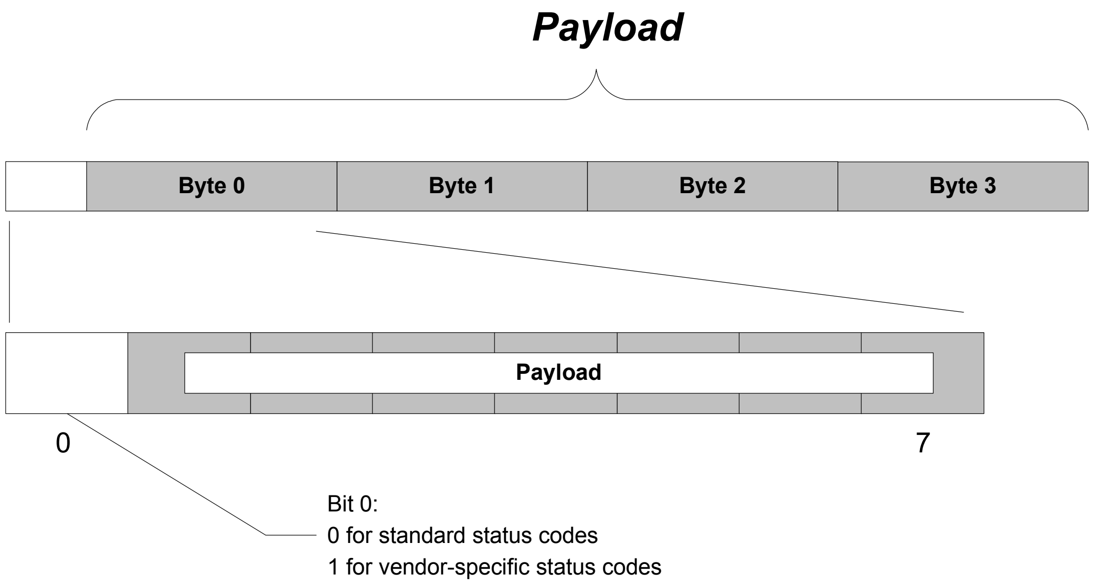
Payload
Byte 0
Byte 1
Byte 2
Byte 3
Payload
0
7
Bit 0:
0 for standard status codes
1 for vendor-specific status codes

Figure 4 – XAM status type diagram

Note: Success is denoted with a status set to zero (bit 0 set to zero because it is a standard status code, and the payload for success uses the standard value of 0).

5.2.3 Error conditions

This document describes a list of error conditions that are associated with each method. However, this list is not a complete list of all possible errors; instead, it is a list of standardized errors. The specification does not limit errors to those standard errors described in this text. For example, a VIM is likely to generate errors that are specific to the related XAM Storage System. Applications should be prepared to handle these non-standard exceptional conditions.

A method is defined to get an error token from a XAM status type. An error token is in the form of a string. The string starts with a prefix ("xam" for standard errors or the reverse DNS of the vendor for non-standard errors) followed by a separator ("/") and ending with a non-localized UTF-8 substring that briefly describes the error. For example, a standard "out of memory" error might generate the following error token:

```
"xam/out of memory"
```

This method requires an XSystem or a XAM Library object. If a XAM Library object is used, the method will not be able to generate vendor-specific error tokens. Such cases will result in the following error token:

```
"xam/unknown error"
```

Note: If an XSystem handle is used, the XSystem does not need to be authenticated for the method to work.

5.2.4 XAM handles

5.2.4.1 XSets, XSystems, and XAM – objects with fields

Field access methods in XAM are scoped to specific objects (XSets or XSystems) or are global in scope (XAM Library). To provide a constant type for all of these objects, handles are used. All of these types inherit from a single common root, the "xam_handle_t", as described below:

```
typedef xam_int xam_handle_t;
```

This type is defined in the appropriate header file. The types for XSets and XSystems use this syntax as their base, as follows:

```
typedef xam_handle_t xset_handle;
typedef xam_handle_t xsystem_handle;
```

These types are also defined in the appropriate header file.

The globally scoped 'xam library' handle has no constructor or destructor. A special value is assigned to this handle that may not be used for any other handle. The xam_library_handle is treated as a global scope reference, in that using this special value should always be interpreted as referring to the XAM Library object. The constant defined for this purpose in xam.h is shown below:

```
#define XAM_LIBRARY_HANDLE (xam_handle_t)1;
```

An error occurs when a NULL value is passed to any method that expects a handle.

5.2.4.2 XIterator

An XIterator is used to enumerate the field names of the fields on an XSet, XSystem, or XAM object. It does not have fields itself; therefore, it is not a xam_handle type. The 'xiterator_handle' is defined in xam_types.h, as below:

```
typedef xam_int xiterator_handle;
```

The XIterator can be created with a prefix (in which case only those fields that match the prefix are enumerated) or without one (in which case all fields are listed). Methods also exist to retrieve the next field name (and advance the cursor) and to release the resources associated with the handle. The specific methods associated with an XIterator are listed with other methods.

5.2.4.3 XStream

An XStream is used to manipulate the value of fields that do not have a simple MIME type. As such, it does not have fields itself and, thus, is not a xam_handle type. The ‘xstream_handle’ is defined in xam_types.h as defined below:

```
typedef xam_int xstream_handle;
```

The XStream uses POSIX-like semantics to manipulate the stream data. An XStream can be opened for reading, writing, or appending; this creates an XStream instance. An XStream instance that is opened for reading can then have blocks of data read from it, while an XStream instance that is opened for writing or appending can have blocks of data written into it. Note that each read or write moves the cursor to the end of the block of data written or read. The location of the cursor can be discovered (tell) and can also be set (seek). Seek is only available on XStreams opened for reading.

5.2.4.4 XAsync

An XAsync is used to track the forward progress and retrieve the results for an asynchronous operation. As such, it does not have fields itself and is not a xam_handle type. The ‘xasync_handle’ is defined in xam_types.h as defined below:

```
typedef xam_int xasync_handle;
```

XAM defines asynchronous versions of synchronous methods that are on the data path and could potentially block for an extended period of time. The XAsync is automatically created if the asynchronous version of a method is called. Depending on the method, and to manage the pending operation, the resultant instance is attached to either a XAM, an XSystem, or an XSet instance. While the operation is pending, the application can query to see if the operation is complete and optionally can halt the operation. The application can use the asynchronous method in one of two ways. It can either register a callback method to be called when the operation completes, or it can poll periodically until the operation has completed. Once the operation has completed, the application can query the XAsync instance for the operation results.

5.2.5 XOPID

Every asynchronous method takes as an input argument a XAM asynchronous operation identifier (XOPID). It can be retrieved from either a pending or completed asynchronous operation. The XOPID type is as defined below:

```
typedef xam_int XOPID;
```

The XOPID is intended to provide a fast mechanism for the application to retrieve its state associated with the asynchronous operation. Because the 64-bit value is specified by the application and is opaque to the XAM Storage System, the application can attach any meaning to it that it wishes, including an index into an application’s data structure, a pointer, or a bitfield.

5.2.6 Callbacks

Every asynchronous method takes, as an optional input argument, a callback method. The callback method will be called when the operation completes (either successfully or unsuccessfully). The XAM callback type is a defined below:

```
typedef
void
(*xasync_callback) (const xasync_handle inHandle);
```

The callback method is defined by the application. Within the callback routine, the XAM application should first retrieve the status of the operation. If the operation was successful, the XAM application can also retrieve the output arguments, using the appropriate methods. It can also retrieve the XOPID to help retrieve the application state that is associated with the asynchronous operation.

5.3 Methods

This section contains a complete list of the methods contained in the API. Note that some error conditions will affect all methods and are not specifically included in each description (e.g., authentication errors when the XSystem instance's authentication expires).

5.3.1 Error token generation

5.3.1.1 XAM_GetErrorToken

Syntax prototype:

```
Xam_boolean
XAM_GetErrorToken (const xam_handle_t inHandle,
                   const xam_status inStatus,
                   xam_string* const outToken);
```

Parameters:

- inHandle is a valid xam_handle containing an XSystem or a XAM Library object reference.
- inStatus is a valid xam_status.
- outToken is a reference to valid storage for a xam_string. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle or xam_handle.
- The second argument is not a valid xam_status.
- The third argument is NULL.

Description:

This method will generate an error token from the xam_status. If passed an XSystem reference, it will be able to generate error tokens for non-standard status. Otherwise, non-standard status will always generate the "xam/unknown error" token.

This method does not require a passed-in XSystem to be authenticated. It will also work on an XSystem that is in a corrupted or aborted state. It returns TRUE on success and FALSE on failure.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.2 Field iteration

The names of all fields (or some fields) that exist on a given XSet, XSystem, or XAM Library object can be enumerated. Note that this field iteration is performed without regard to the type of field.

5.3.2.1 XAM_OpenFieldIterator

Syntax prototype:

```
xam_status
XAM_OpenFieldIterator (const xam_handle_t inHandle,
                       const xam_string inPattern,
                       xiterator_handle* const outIterator);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the fields to be enumerated.
- inPattern is a valid xam_string containing a valid, NULL terminated UTF-8 byte sequence. The pattern in this xam_string will be used to filter the fields which will be enumerated. Those fields that do not belong with the specified pattern will not be included in the enumeration. The pattern is very simple: the byte sequence is treated as an explicit prefix, and if the beginning of a field name does not match the exact bit sequence of the specified pattern, it will be filtered out of the results. All fields are considered to begin with an empty string; thus, specifying an empty string in the pattern will result in no fields being filtered.
- outIterator is a reference to valid storage for an xiterator_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid prefix (invalid UTF-8).
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method acts as a factory interface, creating an XIterator from an XSet, XSystem, or XAM object (e.g., objects that contain fields). This iterator is used to discover the field names of fields on the object in scope (e.g., an XSet, XSystem, or XAM object). Only those fields whose names begin with the distinct bit sequence as specified in the pattern will be included in the enumeration.

Resources associated with the XIterator must be explicitly released. Once the resources are released, the XIterator will no longer be valid.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.2.2 XIterator_Next

Syntax prototype:

```
xam_status
XIterator_Next (const xiterator_handle inHandle,
                xam_string* const outName);
```

Parameters:

- inHandle is a valid xiterator_handle.
- outName is a reference to valid storage for a xam_string. The result is the name of the field following the current cursor (e.g., the field name of the field at the current cursor/position in the iteration). The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xiterator_handle.
- The second argument is NULL.
- Undefined errors will occur if the resources associated with the XIterator have already been released.

Description:

This method copies the field name of the field at the current cursor of the iteration into the provided storage. The cursor is then advanced to the next field. On reading past the last field, an empty string will be returned.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.2.3 XIterator_HasNext

Syntax prototype:

```
xam_status
XIterator_HasNext (const xiterator_handle inHandle,
                   xam_boolean* const outHasNext);
```

Parameters:

- inHandle is a valid xiterator_handle.
- outHasNext is a reference to valid storage for a xam_boolean. It is set to TRUE if there are more fields following the current cursor (e.g., after the field at the current cursor/position in the iteration). The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xiterator_handle.
- The second argument is NULL.
- Undefined errors will occur if the resources associated with the XIterator have already been released.

Description:

This method indicates if there are fields following the field at the current cursor of the iteration into the provided storage.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.2.4 XIterator_Close

Syntax prototype:

```
xam_status
XIterator_Close (xiterator_handle inHandle);
```

Parameters:

- inHandle is a valid xiterator_handle.

Error conditions:

- The first argument is not a valid xiterator_handle.
- Undefined errors will occur if the resources associated with the XIterator have already been released.

Description:

This method releases the resources associated with an open XIterator. After this method is called, the XIterator may no longer be used.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3 Field manipulation

While all fields are typed using MIME types, these types are divided into two separate categories: those with a MIME type in the stype set (properties) and those that are not (XStreams). Some field methods can be used for any type of field (the generic field methods), some can be used only to operate on properties, and the remainder can be used only to operate on XStreams.

5.3.3.1 Generic field methods

5.3.3.1.1 XAM_ContainsField

Syntax prototype:

```
xam_status
XAM_ContainsField (const xam_handle_t inHandle,
                   const xam_string inName,
                   xam_boolean* const outContained);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM Library object reference.
- inName is a xam_string containing the name of the field.
- outContained is a reference to valid storage for a xam_boolean. It is set to TRUE if the field is contained in the XSet, XSystem, or XAM Library. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will set the provided boolean to TRUE if the field is contained in the XSet, XSystem, or XAM Library. Otherwise, it will be set to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.2 XAM_SetFieldAsBinding

Syntax prototype:

```
xam_status
XAM_SetFieldAsBinding (const xset_handle inHandle,
                       const xam_string inName);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will set the binding attribute of a field to TRUE. Note that unlike the other field methods, this method can only be used with XSets.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.3 XAM_SetFieldAsNonbinding

Syntax prototype:

```
xam_status
XAM_SetFieldAsNonbinding (const xset_handle inHandle,
                          const xam_string inName);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will set the binding attribute of a field to FALSE. Note that unlike the other field methods, this method can only be used with XSets.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.4 XAM_GetFieldType

Syntax prototype:

```
xam_status
XAM_GetFieldType (const xam_handle_t inHandle,
                  const xam_string inName,
                  xam_string* const outType);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outType is a reference to valid storage for a xam_string. The result is the MIME type of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will copy the MIME type of the named field into the provided xam_string.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.5 XAM_GetFieldLength

Syntax prototype:

```
xam_status
XAM_GetFieldLength (const xam_handle_t inHandle,
                    const xam_string inName,
                    xam_int* const outLength);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outLength is a reference to valid storage for a xam_int. The result is the number of bytes of the value of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will copy the length of the named field into the provided xam_int.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.6 XAM_GetFieldBinding

Syntax prototype:

```
xam_status
XAM_GetFieldBinding (const xam_handle_t inHandle,
                     const xam_string inName,
                     xam_boolean* const outBinding);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outBinding is a reference to valid storage for a xam_boolean. The result is TRUE if the binding attribute of the named field is TRUE or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will set the xam_boolean value to TRUE if the binding attribute of the named field is TRUE or to FALSE otherwise.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.7 XAM_GetFieldReadOnly

Syntax prototype:

```
xam_status
XAM_GetFieldReadOnly (const xam_handle_t inHandle,
                      const xam_string inName,
                      xam_boolean* const outReadOnly);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outReadOnly is a reference to valid storage for a xam_boolean. The result is TRUE, if the readonly attribute of the named field is TRUE, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will set the xam_boolean value to TRUE, if the readonly attribute of the named field is TRUE, or to FALSE otherwise.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.1.8 XAM_DeleteField

Syntax prototype:

```
xam_status
XAM_DeleteField (const xam_handle_t inHandle,
                 const xam_string inName);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object contains the named field.
- inName is a xam_string containing the name of the field to delete.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the name refers to a binding field.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will remove a field from the XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2 Property field methods

5.3.3.2.1 XAM_CreateBoolean

Syntax prototype:

```
xam_status
XAM_CreateBoolean (const xam_handle_t inHandle,
                   const xam_string inName,
                   const xam_boolean inBinding,
                   const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_boolean containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.boolean” on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.2 XAM_CreateInt

Syntax prototype:

```
xam_status
XAM_CreateInt (const xam_handle_t inHandle,
               const xam_string inName,
               const xam_boolean inBinding,
               const xam_int inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_int containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.int” on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.3 XAM_CreateDouble

Syntax prototype:

```
xam_status
XAM_CreateDouble (const xam_handle_t inHandle,
                  const xam_string inName,
                  const xam_boolean inBinding,
                  const xam_double inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_double containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.double” on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.4 XAM_CreateXUID

Syntax prototype:

```
xam_status
XAM_CreateXUID (const xam_handle_t inHandle,
                const xam_string inName,
                const xam_boolean inBinding,
                const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_xuid containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The format of the fourth argument is not valid (i.e., not a valid xuid format).
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.xuid” on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.5 XAM_CreateString

Syntax prototype:

```
xam_status
XAM_CreateString (const xam_handle_t inHandle,
                  const xam_string inName,
                  const xam_boolean inBinding,
                  const xam_string inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_string containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.

- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to "application/vnd.snia.xam.string" on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.6 XAM_CreateDatetime

Syntax prototype:

```
xam_status
XAM_CreateDatetime (const xam_handle_t inHandle,
                    const xam_string inName,
                    const xam_boolean inBinding,
                    const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_datetime containing the value to be stored.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is true, and the handle refers to an xsystem or xam library object.
- The format of the fourth argument is not valid (i.e., not a valid datetime format).
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is being created as binding.

- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.datetime” on the object referenced by the passed-in xam_handle_t. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.7 XAM_SetBoolean

Syntax prototype:

```
xam_status
XAM_SetBoolean (const xam_handle_t inHandle,
                const xam_string inName,
                const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_boolean containing the new value to be stored.

Error conditions:

- The named field is not of type boolean.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.

- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.boolean” on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.8 XAM_SetInt

Syntax prototype:

```
xam_status
XAM_SetInt (const xam_handle_t inHandle,
            const xam_string inName,
            const xam_int inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_int containing the new value to be stored.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.

- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.int” on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.9 XAM_SetDouble

Syntax prototype:

```
xam_status
XAM_SetDouble (const xam_handle_t inHandle,
               const xam_string inName,
               const xam_double inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_double containing the new value to be stored.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.

- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.double” on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.10XAM_SetXUID

Syntax prototype:

```
xam_status
XAM_SetXUID (const xam_handle_t inHandle,
             const xam_string inName,
             const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_xuid containing the new value to be stored.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The format of the third argument is not valid (i.e., not a valid XUID format).
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.

- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.xuid” on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.11 XAM_SetString

Syntax prototype:

```
xam_status
XAM_SetString (const xam_handle_t inHandle,
               const xam_string inName,
               const xam_string inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_string containing the new value to be stored.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.

- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.string” on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.12XAM_SetDatetime

Syntax prototype:

```
xam_status
XAM_SetDatetime (const xam_handle_t inHandle,
                 const xam_string inName,
                 const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_datetime containing the new value to be stored.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The format of the third argument is not valid (i.e., not a valid datetime format).
- The xam_handle_t contains an XSet was opened in readonly mode.

- The xam_handle_t contains an XSet was opened in restricted mode and the field is a binding field.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to "application/vnd.snia.xam.datetime" on the object referenced by the passed-in xam_handle_t. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.13XAM_GetBoolean

Syntax prototype:

```
xam_status
XAM_GetBoolean (const xam_handle_t inHandle,
                const xam_string inName,
                xam_boolean* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_boolean. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type boolean.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.

- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.boolean” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.14XAM_GetInt

Syntax prototype:

```
xam_status
XAM_GetInt (const xam_handle_t inHandle,
            const xam_string inName,
            xam_int* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_int. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.

- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.int” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.15XAM_GetDouble

Syntax prototype:

```
xam_status
XAM_GetDouble (const xam_handle_t inHandle,
               const xam_string inName,
               xam_double* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_double. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.

- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.double” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.16XAM_GetXUID

Syntax prototype:

```
xam_status
XAM_GetXUID (const xam_handle_t inHandle,
             const xam_string inName,
             xam_xuid* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_xuid. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.xuid” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.17 XAM_GetString

Syntax prototype:

```
xam_status
XAM_GetString (const xam_handle_t inHandle,
               const xam_string inName,
               xam_string* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_string. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.string” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.2.18 XAM_GetDatetime

Syntax prototype:

```
xam_status
XAM_GetDatetime (const xam_handle_t inHandle,
                 const xam_string inName,
                 xam_datetime* const outValue);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_datetime. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.datetime” on the object referenced by the passed-in xam_handle_t.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.3 XStream field methods

5.3.3.3.1 XAM_CreateXStream

Syntax prototype:

```
xam_status
XAM_CreateXStream (const xam_handle_t inHandle,
                   const xam_string inName,
                   const xam_boolean inBinding,
                   const xam_string inType,
                   xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new XStream field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inType is a xam_string that contains the MIME type of the field.
- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The fourth argument contains an empty string (“” is not a valid MIME type).
- The fourth argument contains an stype.
- The fifth argument is NULL.
- The xam_handle_t contains an XSet that was opened in readonly mode.

- The xam_handle_t contains an XSet that was opened in restricted mode and the field being created is a binding field.
- The xam_handle_t contains an XSet that was opened in restricted mode and is on hold.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSet and the maximum number of XStream fields allowed on this XSet has been reached.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create an XStream field with a type set to the user-defined MIME type on the object referenced by the passed-in xam_handle_t. Its name, MIME type, and binding attributes will be set according to the user-provided parameters. The XStream field is opened in writeonly mode.

Note: The value is not set by the method. This method will create an XStream with a length of zero; other methods must be used to add data to this field. Also, if the xam_handle_t contains an XSet, this method may fail with an error if the maximum number of fields supported on an XSet is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate *.xsystem.limits.maxFieldsPerXSet* on the XSystem instance. For more information on this topic, please consult the [XAM-ARCH].

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.3.2 XAM_OpenXStream

Syntax prototype:

```
xam_status
XAM_OpenXStream (const xam_handle_t inHandle,
                 const xam_string inName,
                 const xam_string inMode,
                 xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inMode is a string indicating the mode to open the XStream in:

- — readonly: open for reading. Write methods will fail on the XStream instance.
- — writeonly: open for writing. Truncates existing data in the XStream. Read and seek methods will fail on the XStream instance.
- — appendonly: open for writing. Appends to existing data in the XStream. Read and seek methods will fail on the XStream instance.

- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument contains anything other than writeonly, appendonly or readonly.
- The fourth argument is NULL.
- The xam_handle_t contains an XSet that was opened in readonly mode, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, the field is binding, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, is on hold, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will create an open XStream in either readonly, writeonly or appendonly mode, based on the mode argument.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete. For applications that wish to use a non-blocking version of this method, refer to XSystem_AsyncOpenXStream.

5.3.3.3.3 XStream_Read

Syntax prototype:

```
xam_status
XStream_Read (const xstream_handle inHandle,
              char* const ioBuffer,
              const xam_int inBufferLength,
              xam_int* const outBytesRead);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in read mode.
- ioBuffer is a byte array to read the data into.
- inBufferLength is a xam_int set to the number of bytes in the buffer.
- outBytesRead is a reference to valid storage for a xam_int. On return, this value will contain the actual number of bytes read. This value will be less than or equal to the inBufferLength. When there is no more data to be read, a value of -1 will be set. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened writeonly.
- The second argument is NULL.
- The buffer length is less than or equal to zero.

CAUTION: If the inBufferLength is set to a size larger than the actual number of bytes of storage available in the ioBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method transfers data from the storage system into the target buffer, up to the number of bytes requested.

Concurrency requirements:

This method is thread safe.

Blocking:

This method does not block until data is completely read, but will indicate the amount of data that was read in each call. Subsequent calls may be needed to read the remainder of the data. For applications that wish to use a completely non-blocking version of this method, refer to XStream_AsyncRead.

5.3.3.3.4 XStream_Write

Syntax prototype:

```
xam_status
XStream_Write (const xstream_handle inHandle,
               const char* const inBuffer,
               const xam_int inByteCount,
               xam_int* const outByteWritten);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in writeonly mode.
- inBuffer is a byte array containing the data to be written.
- inByteCount is a xam_int set to the number of bytes in the buffer to be written.
- outBytesWritten is a reference to valid storage for a xam_int. On return, this will contain the actual number of bytes written, which will be less than or equal to the inByteCount. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened readonly.
- The second argument is NULL.
- The maximum length (in bytes) of an XStream is exceeded.

CAUTION: If the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method transfers data from the source buffer to the XAM Storage System, up to the number of bytes requested.

Note: This method may fail with an error if the maximum number of bytes supported in an XStream is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate the *.xsystem.limits.maxSizeOfXStream* field on the XSystem instance. For more information on this topic, please consult the [XAM-ARCH].

Concurrency requirements:

This method is thread safe.

Blocking:

This method does not block until all the data in the buffer is completely written, but it will indicate the amount of data that was written in each call. Subsequent calls may be needed to write all of the data. For applications that wish to use a completely non-blocking version of this method, refer to XStream_AsyncWrite.

5.3.3.3.5 XStream_Seek

Syntax prototype:

```
xam_status
XStream_Seek (const xstream_handle inHandle,
              const xam_int inOffset,
              const xam_int inWhence);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in read mode.
- inOffset is a xam_int containing the number of bytes to change the position by.
- inWhence is a xam_int containing a 0, 1, or 2 (indicating where the offset should be measured from). These are defined as follows:
  - 0: The offset is measured from the start of the XStream.
  - 1: The offset is measured from the current position in the XStream.
  - 2: The offset is measured from the end of the XStream

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened writeonly.
- The second and third arguments would result in a new position before to the first byte or past the final byte in the XStream.
- The third argument contains a value other than 0, 1, or 2.

Description:

This method sets the position indicator for the XStream. The new position, measured in bytes, is obtained by adding inOffset bytes to the position specified by inWhence. If inWhence is set to 0, 1, or 2, then the offset is relative to the start of the XStream, the current position, or end-of-data, respectively.

Note: This method can only be used for XStreams opened for read. In addition, this method cannot be used to create sparse files. It is an error to seek past the end of the data in the XStream, as indicated by the field attribute ‘length’.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.3.6 XStream_Tell

Syntax prototype:

```
xam_status
XStream_Tell (const xstream_handle inHandle,
              xam_int* const outPosition);
```

Parameters:

- inHandle is an xstream_handle.
- outPosition is a xam_int containing the position in the XStream.

Error conditions:

- The first argument is not a valid xstream_handle.
- The second argument is NULL.

Description:

This method gets the current value of the XStream position indicator.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.3.7 XStream_Abandon

Syntax prototype:

```
xam_status
XStream_Abandon (const xstream_handle inHandle);
```

Parameters:

- inHandle is an xstream_handle.

Error conditions:

- The first argument is not a valid xstream_handle.

CAUTION: If the XStream has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XStream in its normal state will generate an error when an application attempts to close it, if there are open asynchronous operations being performed on it. Making this call will change the state of the XStream and allow it to be closed without regard for any open asynchronous operations. Note that the XStream will no longer be usable after this call is made, and the only call that will succeed is XStream.close.

CAUTION: This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open asynchronous operations and close the asynchronous operations properly, as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.3.3.8 XStream_Close

Syntax prototype:

```
xam_status
XStream_Close (xstream_handle inHandle);
```

Parameters:

- inHandle is an xstream_handle.

Error conditions:

- The first argument is not a valid xstream_handle.
- The XStream instance was used for import and an import error occurred.

CAUTION: Closing an already closed XStream can produce undefined results, which may include data loss and data corruption.

Description:

This method closes a previously opened XStream. Any resources that were allocated can be released at this point.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete. For applications that wish to use a non-blocking version of this method, refer to XStream_AsyncClose.

5.3.4 Connection administration for a XAM Storage System

5.3.4.1 XAMLibrary_Connect

Syntax prototype:

```
xam_status
XAMLibrary_Connect (const xam_string inXRI,
                    xsystem_handle* const outHandle);
```

Parameters:

- inXRI is a xam_string. It contains the XSystem's Internationalized Resource Identifier. The format of the XRI is listed below:

  ```
  snia-xam://[vimname!]xsystemname[?param=value[{&param=value}]]
  ```

  The vimname is a string that describes which VIM to use, and if it is not specified, the XAM system will choose a VIM to use. A vimname is not allowed to contain a '!' character. The xsystemname is vendor specific; it may be an IP address or some other id. It may not contain '/', '?', or '!' characters. Finally, param'='value pairs can be specified. Note that the '&' character is not permitted in the name/value pair. The full BNF of this format can be found in the XAM Architecture Specification [XAM-ARCH].

- outHandle is a reference to valid storage for an xsystem_handle. On return, this will contain the XSystem handle that was created. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid XRI.
- The second argument is NULL.
- A problem exists with the underlying XAM Storage System or its infrastructure (e.g., a damaged cable for IP attached storage).

Description:

XAM applications connect to XAM Storage Systems by calling this method and specifying the XSystem's Internationalized Resource Identifier (XRI) string as its parameter. It is expected that the XRI will be specified by the local storage system administrators, and applications should strive to make this easily configured at run time.

Note: The XSystem instance is not fully usable until it has been authenticated.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.4.2 XSystem_Authenticate

Syntax prototype:

```
xam_status
XSystem_Authenticate (const xsystem_handle inHandle,
                      const char* const inBuffer,
                      const xam_int inByteCount,
                      xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xsystem_handle.
- inBuffer: Data that is being passed to the authentication mechanism is passed in this array of bytes.
- inByteCount: The number of significant bytes in the passed-in buffer.
- outXStream is a reference to valid storage for an xstream_handle. On return, this will contain the XStream handle that was created, and which contains the XSystem's response to the authentication information. The value that is passed in is not used and is overwritten with the result.

Note: The outXStream must be closed when the application has finished its authentication processing.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The fourth argument is NULL.
- There is an authentication failure.

Note: If the XSystem has been closed or if the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method allows an application to authenticate an XSystem instance. It provides a generic interface to exchange data as part of the authentication process. The application should check for XSystem instance properties with the prefix of *.xsystem.auth.SASLmechanism.list.* to determine which patterns of authentication (mechanisms) are available for use. After a pattern is selected, the appropriate sequence of data exchanges should be made (using this call) in order to authenticate. A failed authentication will make the XSystem instance unusable; applications cannot repeat failed authentications using the same XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.4.3 XSystem_Close

Syntax prototype:

```
xam_status
XSystem_Close (const xsystem_handle inHandle);
```

Parameters:

- inHandle is an xsystem_handle.

Error conditions:

- The first argument is not a valid xsystem_handle.
- There are open XSets or XStreams.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is called to release any resources associated with an XSystem. After calling this method, the closed XSystem should not be used.

Note: This call will fail if there are any open XSets associated with this XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.4.4 XSystem_Abandon

Syntax prototype:

```
xam_status
XSystem_Abandon (const xsystem_handle inHandle);
```

Parameters:

- inHandle is an xsystem_handle.

Error conditions:

- The first argument is not a valid xsystem_handle.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XSystem, in its normal state, will generate an error when an application attempts to close it, if it has open XSets in it. Making this call will change the state of the XSystem and allow it to be closed without regard for any open XSets. Note that the XSystem will no longer be usable after this call is made, and the only call that will succeed is XSystem.close.

CAUTION: This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open XSets and close the XSets properly as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.5 XSet instance creation

5.3.5.1 XSystem_CreateXSet

Syntax prototype:

```
xam_status
XSystem_CreateXSet (const xsystem_handle inHandle,
                    const xam_string inMode,
                    xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inMode is a string indicating the mode to open the XSet in:
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- outXSet is a reference to valid storage for an xset_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is NULL.

- The second argument is not restricted or unrestricted.
- The third argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will create a new, empty XSet instance associated with the XSystem. This XSet will not exist on the XSystem unless that XSet instance is committed.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.5.2 XSystem_OpenXSet

Syntax prototype:

```
xam_status
XSystem_OpenXSet (const xsystem_handle inHandle,
                  const XUID inXUID,
                  const xam_string inMode,
                  xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be opened.
- inMode is a string indicating the mode to open the XSet in:
  - — readonly: open for reading. Adding, deleting, or modifying fields is not allowed. Commit of the XSet instance will fail.
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- outXSet is a reference to valid storage for a xset_handle. On return, this value will contain the XSet handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not readonly, restricted, or unrestricted.
- The XSet is on hold, and the mode is not readonly.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an XSet in the XSystem, returning a handle to an XSet instance associated with the XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete. For applications that wish to use a non-blocking version of this method, refer to XSystem_AsyncOpenXSet.

5.3.5.3 XSystem_CopyXSet

Syntax prototype:

```
xam_status
XSystem_CopyXSet (const xsystem_handle inHandle,
                  const XUID inXUID,
                  const xam_string inMode,
                  xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be copied.
- inMode is a string indicating the mode to open the copied XSet in:
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not

allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.

— unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).

- outXSet is a reference to valid storage for a xset_handle. On return, this value will contain the XSet handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not restricted or unrestricted.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will create a copy of an XSet in the XSystem, returning a handle to an XSet instance associated with the XSystem. This XSet will not exist on the XSystem unless that XSet instance is committed.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete. For applications that wish to use a non-blocking version of this method, refer to XSystem_AsyncCopyXSet.

5.3.6 XSet administration

5.3.6.1 XSystem_IsXSetRetained

Syntax prototype:

```
xam_status
XSystem_IsXSetRetained (const xsystem_handle inHandle,
                        const xam_xuid inXUID,
                        xam_boolean* const outIsRetained);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- outIsRetained is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE, if the XSet is accessible, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will evaluate all retention criteria that exists on the specified XSet and shall return TRUE if there exists retention criterion which would prevent XSet deletion. The method returns FALSE if the retention criteria are not sufficient to describe a complete retention, if the retention is not enabled, or if the retention criteria are valid but the retention period has passed.

This method does not evaluate the “on-hold” status.

A non-fatal error will be returned if the specified XUID is improperly formatted, does not exist in the XSystem, or if the caller is not authorized to read the XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.6.2 XSystem_DeleteXSet

Syntax prototype:

```
xam_status
XSystem_DeleteXSet (const xsystem_handle inHandle,
                    const xam_xuid inXUID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be deleted.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The XSystem does not have authorization to delete an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will delete an XSet from the XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.6.3 XSystem_HoldXSet

Syntax prototype:

```
xam_status
XSystem_HoldXSet (const xsystem_handle inHandle,
                  const XUID inXUID,
                  const xam_string inHoldID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be held.
- inHoldID is a xam_string that contains the ID to be associated with the hold.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The third argument contains a hold id that is already in use for this XSet.
- The XSystem does not have authorization to hold an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will place an XSet on hold. A held XSet cannot be changed in any way. An XSet may be placed on multiple holds, by using different hold ids.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.6.4 XSystem_ReleaseXSet

Syntax prototype:

```
xam_status
XSystem_ReleaseXSet (const xsystem_handle inHandle,
                     const xam_xuid inXUID,
                     const xam_string inHoldID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be held.
- inHoldID is a xam_string that contains the ID associated with the hold.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The third argument contains a hold id that is not in use for this XSet.
- The XSet is not being held at all.
- The XSystem does not have authorization to release a hold from an XSet.
- The XSet is not held or is not held using the specified hold id.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will release a specific hold on an XSet (associated with the hold id).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.6.5 XSystem_AccessXSet

Syntax prototype:

```
xam_status
XSystem_AccessXSet (const xsystem_handle inHandle,
                    const xam_xuid inXUID,
                    const xam_int inMode,
                    xam_boolean* const outIsAccessible);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- inMode: The value is the bitwise OR of the access 'permissions' to be checked (R_OK for read permission, WU_OK for write-user permission, WS_OK for write-system permission, D_OK for delete, H_OK for hold, RE_OK for retention event, J_OK for job and JC_OK for job commit). In addition, there are composite permissions W_OK (WU_OK|WS_OK), RW_OK (R_OK|W_OK) and ALL_OK (RW_OK|D_OK|H_OK|RE_OK|J_OK|JC_OK).
- outIsAccessible is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE if the XSet is accessible according to the access permissions set by mode, FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument does not contain a valid mode.
- The fourth argument is NULL.
- The XSystem does not have authorization to evaluate the accessibility of an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will check the accessibility of an XSet on the XSystem. It is not an error if the XSet does not exist on the XSystem; such an XSet shall be noted as being inaccessible.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.6.6 XSystem_GetXSetAccessTime

Syntax prototype:

```
xam_status
XSystem_GetXSetAccessTime (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           xam_datetime* const outAccessTime);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- outAccessTime is a reference to valid storage for a xam_datetime. On return, this value will be set to the time at which the XSet was last opened or committed, whichever is the most recent. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The XSystem does not have authorization to evaluate the access time of an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will return the time at which the XSet was last opened or committed, whichever is the most recent.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.7 XSet instance administration

5.3.7.1 XSet_Commit

Syntax prototype:

```
xam_status
XSet_Commit (const xset_handle inHandle,
             xam_xuid* const outXUID);
```

Parameters:

- inHandle is an xset_handle.
- outXUID is a reference to valid storage for a XUID. On return, this value will contain the XUID that was assigned to the XSet by the XAM Storage System. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is NULL.
- The XSystem does not have authorization to commit an XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and one or more binding fields have been created, modified, or deleted, or one or more fields have been changed from binding to nonbinding (or vice versa).
- The XSet is not valid, or has been modified in an invalid way (e.g., a field does not have a valid type).
- The XSet contains a running job (see Section 5.3.10.1, “Jobs”) and the XAM Storage System does not support committing running jobs.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will store an XSet in the XSystem. Note this does not close the XSet, which can still be modified as allowed by the authorization of the XSystem. A XUID will be assigned by the XAM Storage System and this XUID will be returned.

Open XStreams will not cause the commit to fail. Only the data that was successfully written to such XSteams will be committed.

If this is a modified XSet (e.g., an existing XSet was opened and changed), then a new XUID may or may not be assigned by the commit, according to the following rules:

- If only nonbinding fields are edited (created, deleted, or changed), then the XAM Storage System shall not assign a new XUID.
- If any binding fields are edited (created, deleted, or changed), then the XAM Storage System shall assign a new XUID.

Applications that use unrestricted modes should be coded to handle cases where the XUID changes when a modified XSet is committed.

If a management policy has not been applied to the XSet before commit, a default management policy will be applied to the XSet at the time of commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete. For applications that wish to use a non-blocking version of this method, refer to XSet_AsyncCommit.

5.3.7.2 XSet_Close

Syntax prototype:

```
xam_status
XSet_Close (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- There are open XStreams.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is called to release any resources associated with an XSet. After calling this method, the closed XSet should not be used.

Note: This call will fail if there are any open XStreams associated with this XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.7.3 XSet_Abandon

Syntax prototype:

```
xam_status
XSet_Abandon (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XSet in its normal state will generate an error when an application attempts to close it, if there are open XStreams in it. Making this call will change the state of the XSet and allow it to be closed without regard for any open XStreams. Note that the XSet will no longer be usable after this call is made, and the only call that will succeed is XSet.close.

CAUTION: This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open XStreams and close the XStreams properly as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8 XSet management administration

5.3.8.1 Access policy

5.3.8.1.1 XSet_ApplyAccessPolicy

Syntax prototype:

```
xam_status
XSet_ApplyAccessPolicy (const xset_handle inHandle,
                        const xam_boolean inBinding,
                        const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

This method will create or modify a property field with the name of *.xset.access.policy* and a type set to "application/vnd.snia.xam.string" on the object referenced by the passed-in xset_handle. Its value and binding attributes will be set according to the user-provided parameters. This field will be used by the XAM Storage System to determine the policies to use when accessing this XSet.

Note: If an access policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default access policy of the XSystem (i.e., the first string in the *.xsystem.access.policy.list.<name>*).

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.1.2 XSet_ResetAccessFields

Syntax prototype:

```
xam_status
XSet_ResetAccessFields (const xset_handle inHandle);
```

Parameters:

- inHandle is a valid xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will remove all access fields from the XSet.

Note: If an access policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default access policy of the XSystem (i.e., the first string in the *.xsystem.access.policy.list.<name>*).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.2 Base management policy

5.3.8.2.1 XSet_ApplyManagementPolicy

Syntax prototype:

```
xam_status
XSet_ApplyManagementPolicy (const xset_handle inHandle,
                            const xam_boolean inBinding,
                            const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

This method will create or modify a property field with the name of “xam.management.policy” and a type set to “application/vnd.snia.xam.string” on the object referenced by the passed-in xset_handle. Its value and binding attributes will be set according to the user-provided parameters. This field will be used by the XAM Storage System to determine the default policies to use when managing this XSet.

Note: If the base management policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default management policy of the XSystem (i.e. *.xsystem.management.policy.default*).

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.2.2 XSet_ResetManagementFields

Syntax prototype:

```
xam_status
XSet_ResetManagementFields (const xset_handle inHandle);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will remove all management fields from the XSet. This will typically result in a new XSet being created and a new XUID being assigned to this XSet at successful commit.

Note: If the base management policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default management policy of the XSystem (i.e., *.xsystem.management.policy.default*).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3 Retention

5.3.8.3.1 XSet_CreateRetention

Syntax prototype:

```
xam_status
XSet_CreateRetention (const xset_handle inHandle,
                      const xam_boolean inBinding,
                      const xam_string inRetentionID);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain a validly formatted retention identifier.
- The retention identifier already exists in the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The retention identifier is “base”
- The retention identifier is “event” and the binding input parameter is FALSE.

Description:

This method will create a scope to for storing and evaluating retention criteria. It creates a field with a type of “application/vnd.snia.xam.string” and sets the value to the retention id. The field name is formed by appending the retention id to the following prefix: *.xset.retention.list*. Thus, the final format of the name is *.xset.retention.list.<retention id>*. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Creating a binding set of retention criteria will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.2 XSet_SetRetentionEnabledFlag

Syntax prototype:

```
xam_status
XSet_SetRetentionEnabledFlag (const xset_handle inHandle,
                              const xam_string inRetentionID,
                              const xam_boolean inBinding,
                              const xam_boolean inEnabled);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inEnabled is a xam_boolean containing a flag indicating if event retention is enabled on this XSet or not. If the flag is set to TRUE, event retention is enabled; otherwise, it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- Enabled is being set to false after it was set to true.
- The retention identifier is “base”.

Description:

This method will enabled or disable retention that is scoped by the specified retention id. This flag is stored in a field of type “application/vnd.snia.xam.boolean”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.enabled*); thus, the final format of the name is *.xset.retention.<retention id>.enabled*. If the field does not exist, it will be created; otherwise the value will be updated if and only if the value is changed from false to true - if the value is set to true it cannot be changed. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.3 XSet_ApplyRetentionEnabledPolicy

Syntax prototype:

```
xam_status
XSet_ApplyRetentionEnabledPolicy (const xset_handle inHandle,
                                  const xam_string inRetentionID,
                                  const xam_boolean inBinding,
                                  const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The applied policy has the effect of disabling retention for this retention ID after it was previously enabled.
- The retention identifier is “base”.

Description:

This method will enabled or disable retention that is scoped by the specified retention id. The policy name of the policy holding the enabled flag is stored in a field of type “application/vnd.snia.xam.string”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.enabled.policy*); thus, the final format of the name is *.xset.retention.<retention id>.enabled.policy*. If the field does not exist, it will be created; otherwise the value will be updated if and only if the value is changed from false to true - if the value is set to true it cannot be changed. It will have its binding attribute set according to the binding flag that is set by the application.

Note: If the *.xset.retention.<retention id>.enabled* field is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.4 XSet_SetRetentionDuration

Syntax prototype:

```
xam_status
XSet_SetRetentionDuration (const xset_handle inHandle,
                           const xam_string inRetentionID,
                           const xam_boolean inBinding,
                           const xam_int inDuration);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inDuration is a xam_int containing the amount of time (measured in milliseconds from the time of commit) to retain the XSet. Zero indicates no retention, while a negative one (-1) indicates infinite retention.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain a valid duration.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.

- The field already exists on the XSet, and the specified duration value is less than the existing duration value.
- The retention identifier is “base”.

Description:

This method will set the duration of retention that is scoped by the specified retention id. This flag is stored in a field of type “application/vnd.snia.xam.int”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.duration*); thus, the final format of the name is *.xset.retention.<retention id>.duration*. If the field does not exist, it will be created; otherwise the value will be updated if and only if the duration is increased. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.5 XSet_ApplyRetentionDurationPolicy

Syntax prototype:

```
xam_status
XSet_ApplyRetentionDurationPolicy (const xset_handle inHandle,
                                   const xset_string inRetentionID,
                                   const xam_boolean inBinding,
                                   const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.

- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The applied policy has the effect of decreasing the duration for this retention ID.
- The retention identifier is “base”.

Description:

This method will set the duration of retention that is scoped by the specified retention id. This policy name is stored in a field of type “application/vnd.snia.xam.string”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.duration.policy*); thus, the final format of the name is *.xset.retention.<retention id>.duration.policy*. If the field does not exist, it will be created; otherwise the value will be updated if and only if the duration is increased. It will have its binding attribute set according to the binding flag that is set by the application.

Note: If the *.xset.retention.<retention id>.duration* field is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.6 XSet_SetRetentionStarttime

Syntax prototype:

```
xam_status
XSet_SetRetentionStarttime (const xset_handle inHandle,
                            const xam_string inRetentionID,
                            const xam_boolean inBinding);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- This method has already been used on an XSet.
- The retention identifier is “base”.

Description:

This method will set the start time of retention that is scoped by the specified retention id. The current time of the XSystem is stored in a field of type “application/vnd.snia.xam.datetime”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.starttime*); thus, the final format of the name is *.xset.retention.<retention id>.starttime*. If the field does not exist, it will be created; otherwise, an error will be generated, as it is not allowed to change the start time once set. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.7 XSet_SetBaseRetention

Syntax prototype:

```
xam_status
XSet_SetBaseRetention (const xset_handle inHandle,
                        const xam_boolean inBinding,
                        const xam_int inDuration);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.

- inDuration is a xam_int containing the amount of time (measured in milliseconds from the time of commit) to retain the XSet. Zero indicates no retention, while a negative one (-1) indicates infinite retention.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain a valid duration.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.
- The field already exists on the XSet, and the specified duration value is less than the existing duration value.

Description:

If this XSet does not already contain the field *.xset.retention.list.base*, this method will create the field with a type of “application/vnd.snia.xam.string” and set the value to “base”. It will also create the “application/vnd.snia.xam.boolean” field *.xset.retention.base.enabled* and set the value to true. The duration will be stored in a field named *.xset.retention.base.duration*. This field is of type “application/vnd.snia.xam.int”. If the field already exists, its value will be changed to match the passed in duration if and only if the duration of the retention is not reduced; the method will generate an error if the duration is reduced. If the field does not already exist, it will be created with the specified duration as the value. The *.xset.retention.base.duration* field will have its binding attribute set according to the binding flag that is set by the application. The *.xset.retention.list.base* is always a binding field.

These fields will be used by the XAM Storage System to determine the base retention duration to use when managing this XSet.

Note: Changing *.xset.retention.base.duration* from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Note: When an XSet instance containing the field *.xset.retention.list.base* is first committed, the field *.xset.retention.base.starttime* will be created as a binding field and have its value set to *.xset.time.xuid*.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.3.8 XSet_ApplyBaseRetentionPolicy

Syntax prototype:

```
xam_status
XSet_ApplyBaseRetentionPolicy (const xset_handle inHandle,
                               const xam_boolean inBinding,
                               const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not already contain the field *.xset.retention.list.base*, this method will create the field with a type of “application/vnd.snia.xam.string” and set the value to “base”. It will also create the “application/vnd.snia.xam.boolean” field *.xset.retention.base.enabled* and set the value to true. The duration policy will be stored in a field named *.xset.retention.base.duration.policy*. This field is of type “application/vnd.snia.xam.string”. If the field already exists, its value will be changed to match the passed-in policy, only if the policy would not reduce the duration of the retention; the method will generate an error if the policy reduces the duration. If the field does not already exist, it will be created with the specified policy name as the value. These fields will have their binding attribute set according to the binding flag that is set by the application.

These fields will be used by the XAM Storage System to determine the base retention duration to use when managing this XSet.

Note: If the *.xset.retention.base.duration* field is also present on the XSet, it will be used by the XAM Storage System in preference to this policy field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Note: When an XSet instance containing the field *.xset.retention.list.base* is first committed, the field *.xset.retention.base.starttime* will be created and have its value set to *.xset.time.xuid*.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.4 AutoDelete

5.3.8.4.1 XSet_ApplyAutoDeletePolicy

Syntax prototype:

```
xam_status
XSet_ApplyAutoDeletePolicy (const xset_handle inHandle,
                            const xam_boolean inBinding,
                            const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have an auto delete policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.deletion.autodelete.policy* and a type set to “application/vnd.snia.xam.string”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically deleted on expiration of retention.

Note: If this method makes an XSet eligible for deletion, there is no guarantee that the XSet will be deleted before the call returns.

Note: If the *.xset.deletion.autodelete* field is also present on the XSet it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.4.2 XSet_SetAutoDelete

Syntax prototype:

```
xam_status
XSet_SetAutoDelete (const xset_handle inHandle,
                    const xam_boolean inBinding,
                    const xam_boolean inAutoDelete);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inAutoDelete is a xam_boolean containing a flag indicating if autodelete is enabled on this XSet or not. If the flag is set to TRUE, autodelete is enabled; otherwise, it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have auto delete set on it, this method will create a property field on the specified XSet with the name of *.xset.deletion.autodelete* and a type set to “application/vnd.snia.xam.boolean”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically deleted on expiration of retention.

Note: If this method makes an XSet eligible for deletion, there is no guarantee that the XSet will be deleted before the call returns.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.5 Shred

5.3.8.5.1 XSet_ApplyShredPolicy

Syntax prototype:

```
xam_status
XSet_ApplyShredPolicy (const xset_handle inHandle,
                        const xam_boolean inBinding,
                        const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have an auto shred policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.deletion.shred.policy* and a type set to 'application/ vnd.snia.xam.string'. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be shredded after XSet deletion. If the *.xset.deletion.shred* field is also present on the XSet it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.5.2 XSet_SetShred

Syntax prototype:

```
xam_status
XSet_SetShred (const xset_handle inHandle,
               const xam_boolean inBinding,
               const xam_boolean inShred);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inShred is a xam_boolean containing a flag indicating if shredding is enabled on this XSet or not. If the flag is set to TRUE, shredding is enabled, otherwise it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have auto shred set on it, this method will create a property field on the specified XSet with the name of *.xset.deletion.shred* and a type set to "application/vnd.snia.xam.boolean". Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be shredded after deletion.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.6 Storage policy

5.3.8.6.1 XSet_ApplyStoragePolicy

Syntax prototype:

```
xam_status
XSet_ApplyStoragePolicy (const xset_handle inHandle,
                         const xam_boolean inBinding,
                         const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have a storage policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.storage.policy* and a type set to "application/vnd.snia.xam.string". Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine the storage policy of the XSet.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.7 XSet management introspection

5.3.8.7.1 XSet_GetActualRetentionDuration

Syntax prototype:

```
xam_status
XSet_GetActualRetentionDuration (const xset_handle inHandle,
                                  const xam_string inRetentionID,
                                  xam_int* const outDuration);
```

Parameters:

- inHandle is a valid xset_handle.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.
- outDuration is a reference to valid storage for a xam_int. On return, this value will be set to the actual event retention duration (in milliseconds) that is currently in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention identifier does not exist in the XSet.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a retention duration policy that does not exist.

- The XSet instance was imported and contains a retention duration policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect the retention duration that is currently in effect for the XSet under the scope of the specified retention id and return that duration to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.7.2 XSet_GetActualRetentionEnabled

Syntax prototype:

```
xam_status
XSet_GetActualRetentionEnabled (const xset_handle inHandle,
                                const xset_string inRetentionID,
                                xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention identifier does not exist in the XSet.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a retention enabled policy that does not exist.
- The XSet instance was imported and contains a retention enabled policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if retention is enabled for the XSet under the scope of the specified retention id and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.7.3 XSet_GetActualAutoDelete

Syntax prototype:

```
xam_status
XSet_GetActualAutoDelete (const xset_handle inHandle,
                          xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains an auto-delete policy that does not exist.
- The XSet instance was imported and contains an auto-delete policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if auto delete is enabled for the XSet and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.8.7.4 XSet_GetActualShred

Syntax prototype:

```
xam_status
XSet_GetActualShred (const xset_handle inHandle,
                     xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a shred policy that does not exist.
- The XSet instance was imported and contains a shred policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if auto shred is enabled for the XSet and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.9 XSet export and import

5.3.9.1 XSet_OpenExportXStream

Syntax prototype:

```
xam_status
XSet_OpenExportXStream (const xset_handle inHandle,
                                  xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xset_handle.
- outXStream is a reference to valid storage for a xstream_handle. On return, this value will contain the XStream handle of an XStream opened in readonly mode. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSystem does not have authorization to export an XSet.
- The XSet has any open XStreams (including import or export XStreams).
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet has never been committed.
- The XSet has been modified since it was opened.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an export XStream for the XSet. The XSet must have been committed and must not have been modified since it was opened / committed. The XSet will enter an import/export state and will thus generate errors if used for any operation until the export XStream is closed. The data in the original XSet instance will be overwritten.

The XStream will contain a canonical representation of the XSet. This data can be read from the XStream using normal XStream calls and semantics. When the XStream is closed, the XSet will return to a normal state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.9.2 XSet_OpenImportXStream

Syntax prototype:

```
xam_status
```

```
XSet_OpenImportXStream (const xset_handle inHandle,
                        xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xset_handle.
- outXStream is a reference to valid storage for a xstream_handle. On return, this value will contain the XStream handle of an XStream opened in writeonly mode. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSystem does not have authorization to import an XSet.
- The XSet was a not newly created XSet.
- The XSet has been modified since it was created.
- The XSet has any open XStreams (including import or export XStreams).
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an import XStream for the XSet. The XSet will enter an import/export state and will thus generate errors if used for any operation until the XStream is closed. Any data in the original XSet instance will be overwritten.

It is expected that a data stream containing the canonical representation of an XSet will be written into the XStream. When the XStream is closed, the data will be validated. If the data is determined to be valid, then the XSet will return to a normal state (i.e., will no longer generate errors when operated on), but it will now refer to the XSet that was described by the canonical data that was written to the XStream. If the validation of the data fails (i.e., it contains invalid or improperly formatted data), then the XSet will enter a corrupted state. It will no longer be recoverable, and all operations, except XSet.abandon followed by XSet.close, will fail.

After a successful validation, the XSet fields can be examined as any normal fields, and the XSet can be modified. The XSet is not committed, but it is in all ways a normal XSet and may be committed as per normal XSet semantics. If the XSet is committed before any modification to binding fields (adding, modifying, or deleting binding fields, or changing the binding attribute of any fields), then the XUID will be the XUID that is described by the import XStream. Modification to any binding fields, as described above, will result in a new XSet and a new XUID being assigned on a successful commit.

An XSet that is opened in restricted mode does not allow modifications that would result in the creation of a new XSet and assignment of a new XUID on a successful commit; edits that would result in the generation of a new XSet shall result in errors.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.10 Asynchronous operations

5.3.10.1 Jobs

5.3.10.1.1XSet_SubmitJob

Syntax prototype:

```
xam_status
XSet_SubmitJob (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSystem does not have authorization to submit a job.
- The XSet is does not contain valid job control fields.

CAUTION: If The XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will submit a job request to the XAM Storage System. Fields on the XSet will be evaluated as input to the job, according to the semantics of the XAM job control subsystem, which requires the job name to be contained in a string field named *.xam.job.command* (see [XAM-ARCH] for more details). This XSet will be used to communicate health and status information about the job. The status of the job will be contained in the string field *.xam.job.status*. If this field contains a value of “error” then the string field *.xam.job.error* should be evaluated to determine the actual error.

Jobs may use other fields specific to the job in question. In that case, the prefix of the job should be the job command (e.g. if the value of “.xam.job.command” is “.vnd.foo” then all fields used by that job should begin with “.vnd.foo”).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.10.1.2XSet_HaltJob

Syntax prototype:

```
xam_status
XSet_HaltJob (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSystem does not have authorization to halt a job.
- The XSet is does not contain valid job control fields.
- The XSet was not used to submit a job.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will stop a currently running job in the XAM Storage System, if the XSet was used to start a job. Fields on the XSet will be evaluated as input to the job, according to the semantics of the XAM job control subsystem (refer to [XAM-ARCH] for more details).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

5.3.10.2 XSet async I/O

5.3.10.2.1XSystem_AsyncOpenXSet

Syntax prototype:

```
xam_status
XSystem_AsyncOpenXSet (const xsystem_handle inHandle,
                        const XUID inXUID,
                        const xam_string inMode,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                              xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be opened.
- inMode is a string indicating the mode to open the XSet in:
  - readonly: open for reading. Adding, deleting, or modifying fields is not allowed. Commit of the XSet instance will fail.
  - restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not readonly, restricted, or unrestricted.
- The XSet is on hold, and the mode is not readonly.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The sixth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous opening of an XSet in the XSystem, ultimately returning a handle to an XSet instance associated with the XSystem. The specified callback will be invoked as part of the asynchronous opening. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method will not block until complete, and will return control immediately. For applications that wish to use the blocking version of this method, refer to XSystem_OpenXSet.

5.3.10.2.2XSystem_AsyncCopyXSet

Syntax prototype:

```
xam_status
XSystem_AsyncCopyXSet (const xsystem_handle inHandle,
                       const XUID inXUID,
                       const xam_string inMode,
                       const XOPID inXOPID,
                       xasync_callback inCallback,
                       xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be copied.
- inMode is a string indicating the mode to open the copied XSet in:
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.

- The third argument is not restricted or unrestricted.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The sixth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous copying of an XSet in the XSystem, ultimately returning a handle to an XSet instance associated with the XSystem. The specified callback will be invoked as part of the asynchronous copying. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method will not block until complete, and will return control immediately. For applications that wish to use the blocking version of this method, refer to XSystem_CopyXSet.

5.3.10.2.3XAM_AsyncOpenXStream

Syntax prototype:

```
xam_status
XAM_AsyncOpenXStream (const xam_handle_t inHandle,
                      const xam_string inName,
                      const xam_string inMode,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be opened.
- inMode is a string indicating the mode to open the XStream in:
  - — readonly: open for reading. Write methods will fail on the XStream instance.
  - — writeonly: open for writing. Truncates existing data in the XStream. Read and seek methods will fail on the XStream instance.

  - appendonly: open for writing. Appends to existing data in the XStream. Read and seek methods will fail on the XStream instance.
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument contains anything other than writeonly, appendonly or readonly.
- The sixth argument is NULL.
- The xam_handle_t contains an XSet that was opened in readonly mode, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, the field is binding, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, is on hold, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method will begin the asynchronous opening of XStream in either readonly, writeonly, or appendonly mode, based on the mode argument. The specified callback will be invoked as part of the asynchronous opening. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method will not block until complete, and will return control immediately. For applications that wish to use the blocking version of this method, refer to XSystem_OpenXStream.

5.3.10.2.4 XStream_AsyncRead

Syntax prototype:

```
xam_status
XStream_AsyncRead (const xstream_handle inHandle,
                   char* const ioBuffer,
                   const xam_int inBufferLength,
                   const XOPID inXOPID,
                   xasync_callback inCallback,
                   xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in readonly mode.
- ioBuffer is a byte array to read the data into.
- inBufferLength is a xam_int set to the number of bytes in the buffer.
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened writeonly.
- The second argument is NULL.
- The buffer length is less than or equal to zero.
- The sixth argument is NULL.

CAUTION: If the inBufferLength is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous transfer of data from the storage system into the target buffer, up to the number of bytes requested. The specified callback will be invoked as part of the asynchronous transfer. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method returns immediately. For applications that wish to use a blocking version of this method, refer to XStream_Read.

5.3.10.2.5XStream_AsyncWrite

Syntax prototype:

```
xam_status
XStream_AsyncWrite (const xstream_handle inHandle,
                    const char* const inBuffer,
                    const xam_int inByteCount,
                    const XOPID inXOPID,
                    xasync_callback inCallback,
                    xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in writeonly mode.
- inBuffer is a byte array containing the data to be written.
- inByteCount is a xam_int set to the number of bytes in the buffer to be written.
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened readonly.
- The second argument is NULL.
- The maximum length (in bytes) of an XStream is exceeded.
- The sixth argument is NULL.

CAUTION: If the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous transfer of data from the source buffer to the XAM Storage System, up to the number of bytes requested. The specified callback will be invoked as part of the asynchronous transfer. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Note: This method may fail with an error if the maximum number of bytes supported in an XStream is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate the *.xsystem.limits.maxSizeOfXStream* field on the XSystem instance. For more information on this topic, please consult [XAM-ARCH].

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method returns immediately. For applications that wish to use a blocking version of this method, refer to XStream_Write.

5.3.10.2.6XStream_AsyncClose

Syntax prototype:

```
xam_status
XStream_AsyncClose (const xstream_handle inHandleXStream,
                    const XOPID inXOPID,
                    xasync_callback inCallback,
                    xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandleXStream is an xstream_handle.
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The sixth argument is NULL.

CAUTION: Closing an already closed XStream can produce undefined results, including data loss and data corruption.

Description:

This method will begin the asynchronous closing of a previously opened XStream. Any resources that were allocated can be released at this point. The specified callback will be invoked as part of the asynchronous close. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Note: It is the responsibility of the application to track the parent of the XStream. The XOPID can be used for this.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method returns immediately. For applications that wish to use a blocking version of this method, refer to XStream_Close.

5.3.10.2.7XSet_AsyncCommit

Syntax prototype:

```
xam_status
XSet_AsyncCommit (const xset_handle inHandle,
                  const XOPID inXOPID,
                  xasync_callback inCallback,
                  xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xset_handle.
- inXOPID is an application assigned id used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The XSystem does not have authorization to commit an XSet.
- The XSet that was opened in readonly mode.
- The XSet was opened in restricted mode and one or more binding fields have been created, modified, or deleted, or one or more fields have been changed from binding to nonbinding (or vice versa).
- The XSet is not valid, or has been modified in an invalid way (e.g., a field does not have a valid type).
- The XSet contains a running job (see Section Section 5.3.10.1, “Jobs”) and the XAM Storage System does not support committing running jobs.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is an asynchronous version of XSet.commit. See Section 5.3.7.1, "XSet_Commit" for additional information.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method returns immediately. For applications that wish to use a blocking version of this method, refer to `XSet_Commit`.

5.3.10.3 Asynchronous Operations Management

Asynchronous operations are in one of two states: pending and completed. When the operation is first initiated, it is in the pending state. Because the operation has not completed, it is only possible to query whether the operation has completed, retrieve the XOPID that was specified when the operation was initiated, and to halt the operation.

5.3.10.3.1XAsync_Halt

Syntax prototype:

```
xam_status
XAsync_Halt (const xasync_handle inHandle);
```

Parameters:

- inHandle is an xasync_handle.

Error conditions:

- The first argument is not a valid xasync_handle.

Description:

This method stops the execution of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.2XAsync_IsComplete

Syntax prototype:

```
xam_status
XAsync_IsComplete (const xasync_handle inHandle,
                   xam_boolean* const outIsComplete);
```

Parameters:

- inHandle is an xasync_handle.
- outIsComplete is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE if the operation has completed, FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.

Description:

This method retrieves the completed state of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.3XAsync_GetXOPID

Syntax prototype:

```
xam_status
XAsync_GetXOPID (const xasync_handle inHandle,
                 XOPID* const outXOPID);
```

Parameters:

- inHandle is an xasync_handle.
- outXOPID is a reference to valid storage for a XOPID. On return, it is set to the value of the XOPID. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the XOPID of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.4 XAsync_GetStatus

Syntax prototype:

```
xam_status
XAsync_GetStatus (const xasync_handle inHandle,
                  xam_status* const outStatus);
```

Parameters:

- inHandle is an xasync_handle.
- outStatus is a reference to valid storage for a xam_status. On return, this value will be set to the status if the operation has completed. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation has not transitioned to the completed state.
- The operation was programmatically halted.

Description:

This method retrieves the xam_status of the operation associated with the Async instance. It may be used after the operation has transitioned to the completed state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.5XAsync_GetXSet

Syntax prototype:

```
xam_status
XAsync_GetXSet (const xasync_handle inHandle,
                xset_handle* const outXSet);
```

Parameters:

- inHandle is an xasync_handle.
- outXSet is a reference to valid storage for a xam_handle. On return, this value will be set to the xset_handle associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no xset_handle associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xset_handle of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.6XAsync_GetXStream

Syntax prototype:

```
xam_status
XAsync_GetXStream (const xasync_handle inHandle,
                   xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xasync_handle.

- outXStream is a reference to valid storage for a xam_handle. On return, this value will be set to the xstream_handle associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no xstream_handle associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xstream_handle of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.7XAsync_GetXUID

Syntax prototype:

```
xam_status
XAsync_GetXUID (const xasync_handle inHandle,
                xam_xuid* const outXUID);
```

Parameters:

- inHandle is an xasync_handle.
- outXUID is a reference to valid storage for a XUID. On return, this value will be set to the XUID associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no XUID associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xset_handle of the operation associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.8XAsync_GetBytesRead

Syntax prototype:

```
xam_status
XAsync_GetBytesRead (const xasync_handle inHandle,
                     xam_int* const outBytesRead);
```

Parameters:

- inHandle is an xasync_handle.
- outBytesRead is a reference to valid storage for a xam_int. On return, this value will be set to the number of bytes read by the operation, zero if no data has been read or if the operation does not read bytes. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the number of bytes read by the operation associated with the Async instance. Not all operations read bytes, and for those operations it will always be set to zero. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.9XAsync_GetBytesWritten

Syntax prototype:

```
xam_status
XAsync_GetBytesWritten (const xasync_handle inHandle,
                        xam_int* const outBytesWritten);
```

Parameters:

- inHandle is an xasync_handle.
- outBytesWritten is a reference to valid storage for a xam_int. On return, this value will be set to the number of bytes written by the operation, zero if no data has been written or if the operation does not write bytes. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the number of bytes written by the operation associated with the Async instance. Not all operations write bytes, and for those operations it will always be set to zero. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.3.10.3.10XAsync_Close

Syntax prototype:

```
xam_status
XAsync_Close (const xasync_handle inHandle);
```

Parameters:

- inHandle is an xasync_handle.

Error conditions:

- The first argument is not a valid xasync_handle.
- The operation has not transitioned to the completed state.

Description:

This method releases the resources of the operation associated with the Async instance and of the Async instance itself. It may be used after the operation has transitioned to the completed state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

5.4 Fields

This section contains an informative summary of the standard fields used in the C API. For a complete description of all fields and their semantics, refer to the XAM Architecture Specification [XAM-ARCH].

5.4.1 XAM Library fields

Table 2 lists the fields that are available on the XAM Library object:

Table 2 – XAM Library fields

Field name	stype	MIME Type
.xam.apiLevel	xam_string	application/vnd.snia.xam.string
.xam.identity	xam_string	application/vnd.snia.xam.string
.xam.log.append	xam_boolean	application/vnd.snia.xam.boolean
.xam.log.level	xam_string	application/vnd.snia.xam.string
.xam.log.max.rollovers	xam_int	application/vnd.snia.xam.int
.xam.log.max.size	xam_int	application/vnd.snia.xam.int
.xam.log.path	xam_string	application/vnd.snia.xam.string
.xam.log.verbosity	xam_string	application/vnd.snia.xam.string
.xam.vim.list.<name>	xam_string	application/vnd.snia.xam.string

- ***.xam.apiLevel:*** used to indicate which version of the XAM API is implemented (e.g. 1.0.0).
- ***.xam.identity***: indicates the origin of the XAM Library (e.g. org.snia). It is intended for informational use; applications should not code specific behavior with respect to this value.
- ***.xam.log.append:*** indicates whether to append to an existing log file (TRUE) or overwrite (FALSE). The default value is FALSE.
- ***.xam.log.level***: indicates the current level of library logging. This controls what type of information is logged. Applications may set this value to control the log.
- ***.xam.log.max.rollovers:*** indicated the number of previous log files to retain when starting a new log file. The default value is 1.
- ***.xam.log.max.size:*** indicates the maximum size in bytes that a log file may reach before a new log file is started. The default value is 1GB (2^30 = 1,073,741,824 bytes).
- ***.xam.log.path***: indicates the path of the file to write the log into.

- ***.xam.log.verbosity***: The higher the value, the more detail is logged. Applications may set this value to control the log.
- ***.xam.vim.list.<name>***: *.xam.vim.list.* is a prefix for properties listing the names of VIMs that have already been discovered by the XAM Library.

5.4.2 XSystem fields

Table 3 lists the fields that are available on XSystem instances:

Table 3 – XSystem fields

Field name	stype	MIME Type
.xsystem.identity	xam_string	application/vnd.snia.xam.string
.xsystem.time	xam_datetime	application/vnd.snia.xam.datetime
.xsystem.limits.maxFieldsPerXSet	xam_int	application/vnd.snia.xam.int
.xsystem.limits.maxSizeOfXStream	xam_int	application/vnd.snia.xam.int
.xsystem.auth.SASLmechanism.default	xam_string	application/vnd.snia.xam.string
.xsystem.auth.SASLmechanism.list.<mechanism>	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.auth.granule.list.<granule>	xam_string	application/vnd.snia.xam.string
.xsystem.auth.identity.authentication	xam_string	application/vnd.snia.xam.string
.xsystem.auth.identity.authorization	xam_string	application/vnd.snia.xam.string
.xsystem.auth.expiration	xam_datetime	application/vnd.snia.xam.datetime
.xsystem.job.list.<name>	xam_string	application/vnd.snia.xam.string
.xsystem.job.commit.supported	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.job.xam.job.query.continuance.supported	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.job.xam.job.query.level1.supported	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.job.xam.job.query.level2.supported	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.deletion.autodelete	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.deletion.shred	xam_boolean	application/vnd.snia.xam.boolean
.xsystem.management.policy.list.<name>		
.xsystem.management.policy.default		
.xsystem.deletion.autodelete.policy.list.<name>		
.xsystem.deletion.shred.policy.list.<name>		
.xsystem.storage.policy.list.<name>		
.xsystem.retention.duration.policy.list.<name>		
.xsystem.retention.enabled.policy.list.<name>		

- ***.xsystem.identity***: holds the vendor identity of the XSystem instance.

- ***.xsystem.time***: holds the current time of the XSystem instance.
- ***.xsystem.limits.maxFieldsPerXSet:*** holds the maximum number of fields that may be created in an XSet.
- ***.xsystem.limits.maxSizeOfXStream***: holds the maximum size of an XStream.
- ***.xsystem.auth.SASLmechanism.default:*** holds the default SASL mechanism for the connected XSystem.
- ***.xsystem.auth.SASLmechanism.list.<mechanism>***: *.xsystem.auth.SASLmechanism.list.* is a prefix for properties listing the names of supported SASL mechanisms.
- ***.xsystem.auth.granule.list.<granule>***: *.xsystem.auth.granule.list.* is a prefix for properties listing the names of the auth granules.
- ***.xsystem.auth.identity.authentication***: holds the authentication id.
- ***.xsystem.auth.identity.authorization***: holds the authorization id.
- ***.xsystem.job.list.<name>***: *.xsystem.job.list.* is a prefix for properties listing the names of supported jobs.
- ***.xsystem.job.xam.job.query.commit.supported***: TRUE if xsystem supports commits of running queries.
- ***.xsystem.job.xam.job.query.continuance.supported***: TRUE if xsystem supports query that continue while disconnected.
- ***.xsystem.job.job.xam.job.query.level1.supported***: TRUE if xsystem supports level 1 query.
- ***.xsystem.job.job.xam.job.query.level2.supported***: TRUE if xsystem supports level 2 query.
- ***.xsystem.deletion.autodelete***: TRUE if xsystem supports autodelete.
- ***.xsystem.deletion.shred***: TRUE if xsystem supports shred.
- ***.xsystem.management.policy.list.<name>***: a prefix for properties listing the names of management policies.
- ***.xsystem.management.policy.default***: holds the default management policy.
- ***.xsystem.deletion.autodelete.policy.list.<name>***: a prefix for properties listing the names of autodelete policies.
- ***.xsystem.deletion.shred.policy.list.<name>***: a prefix for properties listing the names of shred policies.
- ***.xsystem.storage.policy.list.<name>***:“ a prefix for properties listing the names of storage policies.
- ***.xsystem.retention.duration.policy.list.<name>***: a prefix for properties listing the names of duration policies.
- ***.xsystem.retention.enabled.policy.list.<name>***: a prefix for properties listing the names of event retention policies.

5.4.3 XSet fields

Table 4 lists the fields that are available on all XSet instances:

Table 4 – XSet fields

Field name	stype	MIME Type
.xset.time.creation	xam_datetime	application/vnd.snia.xam.datetime
.xset.time.xuid	xam_datetime	application/vnd.snia.xam.datetime
.xset.time.commit	xam_datetime	application/vnd.snia.xam.datetime
.xset.time.access	xam_datetime	application/vnd.snia.xam.datetime
.xset.time.residency	xam_datetime	application/vnd.snia.xam.datetime
.xset.dirty	xam_boolean	application/vnd.snia.xam.boolean
.xset.xuid	XUID	application/vnd.snia.xam.xuid
.xset.management.policy	xam_string	application/vnd.snia.xam.string
.xset.retention.base.enabled	xam_boolean	application/vnd.snia.xam.boolean
.xset.retention.base.duration	xam_int	application/vnd.snia.xam.int
.xset.retention.base.duration.policy	xam_string	application/vnd.snia.xam.string
.xset.retention.base.starttime	xam_datetime	application/vnd.snia.xam.datetime
.xset.deletion.autodelete.policy	xam_string	application/vnd.snia.xam.string
.xset.deletion.shred.policy	xam_boolean	application/vnd.snia.xam.boolean
.xset.storage.policy	xam_string	application/vnd.snia.xam.string

- ***.xset.time.creation***: holds the time at which the XSet was created.
- ***.xset.time.xuid***: holds the time at which the XUID was assigned to the XSet.
- ***.xset.time.commit***: holds the time at which the XSet was last modified.
- ***.xset.time.access***: holds the time at which the XSet was last opened or committed.
- ***.xset.time.residency*:** holds the time at which the XSet was originally committed.
- ***.xset.dirty***: TRUE when the XSet instance has been modified relative to the XSet.
- ***.xset.xuid**:* holds the XUID of the XSet. If the XSet has not been committed or if a binding modification has been made, this field will not be present.
- ***.xset.management.policy***: determines XSet retention time criteria, autodelete, and shred behavior in the absence of both value and policy management properties.
- ***.xset.retention.base.enabled***: used to determine if the retention information is valid and retention is active. This should always be set to true in an XSet.
- ***.xset.retention.base.duration***: used to determine the value of XSet base retention duration.

- ***.xset.retention.base.duration.policy***: used to determine the value of XSet base retention duration in the absence of the *.xset.retention.base.duration* property.
- ***.xset.retention.base.starttime***: holds the time awhich retention starts.
- ***.xset.deletion.autodelete.policy***: determines the actual value of XSet autodelete in the absence of the .xset.deletion.autodelete property.
- ***.xset.deletion.shred.policy***: determines how XSet and child XStreams are handled after removal.
- ***.xset.storage.policy***: determines how to manage an XSet with respect to storage management capabilities that are outside the scope of XAM, e.g., storage performance, resiliency, and virtualization.

5.4.4 Job fields

Table 5 lists the standard job fields used by all jobs:

Table 5 – Job fields

Field name	stype	MIME Type
.xam.job.command	xam_string	application/vnd.snia.xam.string
.xam.job.status	xam_string	application/vnd.snia.xam.string
.xam.job.error	xam_string	application/vnd.snia.xam.string

- ***.xam.job.command***: holds the job name. Note that this job name is used as a field prefix for all job specific fields
- ***.xam.job.status***: holds the status of the job (either OK or ERROR).
- ***.xam.job.error***: holds the error string of the job in cases where the status is ERROR. This field is not present when the status is OK.

5.4.5 Query job fields

Table 6 lists the fields that are used to control the query job:

Table 6 – Query job fields

Field name	stype	MIME Type
xam.job.query.command	-	text/plain
xam.job.query.level	xam_string	application/vnd.snia.xam.xuid
xam.job.query.results	-	application/vnd.snia.query.xuid_list
xam.job.query.results.count	xam_int	application/vnd.snia.xam.double

- ***xam.job.query.command***: holds the XAM QL string used to run the query.
- ***xam.job.query.level***: holds the query level used at the time the query started.
- ***xam.job.query.results***: an XStream that holds the results of the query.

- ***xam.job.query.results.count***: holds the number of results in the results XStream.

5.5 Using the XAM API – concrete samples

The method sequences below provide example implementations for the abstract samples presented in the previous chapter. These examples are for the purpose of illustrating the sequence of methods, and for clarity ignore such critical concepts as proper error handling.

5.5.1 Create an XSet

This snippet will create an XSet that contains a person's name and a picture of that person. It does not detail how the name, the jpeg buffer and the number of bytes in that buffer are set.

```
xsystem_handle xsys = (xsystem_handle)0;
XAMLibrary_Connect("myXRI", &xsys);
if ( xsys )
{
                xset_handle xset = (xset_handle)0;
                XSystem_CreateXSet(xsys, "unrestricted", &xset);
                if ( xset )
                {
                    // write the name of the person
                    xam_string name; // should contain the name
                    XAM_CreateString(xset, "myName", true, name);

                    // write a buffer containing a jpeg image
                    char* buffer;   // should point to the buffer to write
                    xam_int bcount; // should contain the number of bytes to write
                    xam_int offset = 0;
                    xstream_handle xstream = (xstream_handle)0;
                    XAM_CreateXStream(xset, "myPic", true, "image/jpeg", &xstream);
                    if ( xstream )
                    {
                        xam_int nWritten = 0;
                        while ( bcount > 0 )
                        {
                            XStream_Write(xstream, &buffer[offset], bcount,
                                             &nWritten);
                            bcount -= nWritten;
                            offset += nWritten;
                        }
                        XStream_Close(xstream);
                    }

                    // commit the XSet
                    XUID myxuid;
                    XSet_Commit(xset, &myxuid);

                    // release the resources of the XSet instance
                    XSet_Close(xset);
                }

                // release the resources of the XSystem instance
                XSystem_Close(xsys);
}
```

5.5.2 Create an XSet - alternate asynchronous method

This example will illustrate how an application can specify a callback to asynchronously handle the operation. The callback is outlined below:

```
// global info
xam_status myStatus;
XUID myXUID;

// the callback itself
void
myCallback (const xasync_handle inHandle)
{
             if ( XAsync_IsComplete(inHandle) )
             {
                XAsync_GetStatus(inHandle, &myStatus);
                XAsync_GetXUID(inHandle, &myXUID);
             }
}
```

This callback will be passed as the callback method for the commit. Note that the only difference from the previous example is the commit method itself.

```
xsystem_handle xsys = (xsystem_handle)0;
XAMLibrary_Connect("myXRI", &xsys);
if ( xsys )
{
             xset_handle xset = (xset_handle)0;
             XSystem_CreateXSet(xsys, "unrestricted", &xset);
             if ( xset )
             {
                // write the name of the person
                xam_string name; // should contain the name
                XAM_CreateString(xset, "myName", true, name);

                // write a buffer containing a jpeg image
                char* buffer;   // should point to the buffer to write
                xam_int bcount; // should contain the number of bytes to write
                xam_int offset = 0;
                xstream_handle xstream = (xstream_handle)0;
                XAM_CreateXStream(xset, "myPic", true, "image/jpeg", &xstream);
                if ( xstream )
                {
                   xam_int nWritten = 0;
                   while ( bcount > 0 )
                   {
                      XStream_Write(xstream, &buffer[offset], bcount,
                                    &nWritten);
                      bcount -= nWritten;
                      offset += nWritten;
                   }
                   XStream_Close(xstream);
                }

                // commit the XSet
                xasync_handle async;
                XSet_AsyncCommit(xset, "myCommit", myCallback, &async);

                // release the resources of the XSet instance
                XSet_Close(xset);
             }
```

```
                    // release the resources of the XSystem instance
                    XSystem_Close(xsys);

}
```

5.5.3 Read an XSet

This snippet will open an XSet. It will read fields containing the name and a picture of the named person. It assumes that the appropriate fields are present on the XSet.

```
        xsystem_handle xsys = (xsystem_handle)0;
        XAMLibrary_Connect("myXRI", &xsys);
        if ( xsys )
        {
                    xset_handle xset = (xset_handle)0;
                    XSystem_OpenXSet(xsys, "readonly", &xset);
                    if ( xset )
                    {
                        // read the name of the person
                        xam_string name;
                        XAM_GetString(xset, "myName", name);

                        // read the buffer containing the image
                        char buffer[1000];  // assume image less than 1000 bytes
                        xam_int offset = 0;
                        xam_int buflen = 1000;
                        xstream_handle xstream = (xstream_handle)0;
                        XAM_OpenXStream(xset, "myPic", "readonly", &xstream);
                        if ( xstream )
                        {
                            xam_status stat = (xam_status)0;
                            while ( stat == 0 )
                            {
                                xam_int nRead = 0;
                                stat = XStream_Read(xstream, &buffer[offset], buflen,
                                                    &nRead);
                                if ( nRead == (-1) )
                                    break;
                                else
                                {
                                    offset += nRead;
                                    buflen -= nRead;
                                }
                            }
                            XStream_Close(xstream);
                        }
                        // release the resources of the XSet instance
                        XSet_Close(xset);
                    }

                    // release the resources of the XSystem instance
                    XSystem_Close(xsys);
        }
```

5.5.4 Query an XSet using job methods

This snippet will find the XSet containing the information for J. Smith. It will return the image (note that it assumes that the size of the image is less than 1000 bytes).

```
xsystem_handle xsys = (xsystem_handle)0;
XAMLibrary_Connect(“myXRI”, &xsys);
if ( xsys )
{
            // get the XUID to read the image from
            XUID jsmithXUID = (XUOID)0;
            xset_handle xset = (xset_handle)0;
            XSystem_CreateXSet(xsys, “unrestricted”, &xset);
            if ( xset )
            {
               // create the job command
               xam_string cmd;
               strcpy(cmd, “.xam.job.query”);
               XAM_CreateString(xset, “org.snia.xam.job.command”, true, cmd);

               // create the query specific XAMQL string
               xam_string xamql;
               sprintf(xamql,“SELECT .xset.xuid WHERE myName = /”J. Smith/””);
               XAM_CreateString(xset, “.xam.job.query.command”, true, xamql);

               // submit the job
               XSet_SubmitJob();

               // read the results (assume there will be a XUID)
               char* xbuffer = (char*)jsmithXUID;
               xam_int xbytes = 0;
               xstream_handle xstream = (xstream_handle)0;
               XAM_OpenXStream(xset, “myPic”, “readonly”, &xstream);
               if ( xstream )
               {
                  xam_status stat = (xam_status)0;
                  while ( stat == 0 )
                  {
                     xam_int nRead = 0;
                     stat = XStream_Read(xstream,
                                         &xbuffer[xbytes], 80-xbytes,
                                         &nRead);
                     if ( nRead == (-1) )
                        break;
                     else
                        xbytes += nRead;
                     if ( xbytes >= 80 )
                        break;
                  }
                  XStream_Close(xstream);
               }

               // release the resources of the XSet instance
               XSet_Close(xset);
            }

            // read the image from the xset (assume the XUID read succeeded)
            XSystem_CreateXSet(xsys, “unrestricted”, &xset);
            if ( xset )
            {
               // read the buffer containing the image
               char buffer[1000];  // assume image less than 1000 bytes
               xam_int offset = 0;
               xam_int buflen = 1000;
               xstream_handle xstream = (xstream_handle)0;
```

```
                        XAM_OpenXStream(xset, “myPic”, “readonly”, &xstream);
                        if ( xstream )
                        {
                            xam_status stat = (xam_status)0;
                            while ( stat == 0 )
                            {
                                xam_int nRead = 0;
                                stat = XStream_Read(xstream, &buffer[offset], buflen,
                                                        &nRead);
                                if ( nRead == (-1) )
                                    break;
                                else
                                {
                                    offset += nRead;
                                    buflen -= nRead;
                                }
                            }
                            XStream_Close(xstream);
                        }

                        // release the resources of the XSet instance
                        XSet_Close(xset);
                    }

                    // release the resources of the XSystem instance
                    XSystem_Close(xsys);
        }
```

6 Private (VIM) C API Reference

The private interfaces are defined to allow the XAM Library a standard way to interact with VIMs (Vendor Interface Modules). Applications should avoid coding to these internal C interfaces, as they are intended for use by VIM programmers. The application programmer should view the VIM interfaces as an internal implementation detail; coding to these private APIs will result in non-portable code.

6.1 XAM Library interaction with the VIM

The XAM Library provides the public interfaces intended for use by application programmers. The purpose of the XAM Library is to route the requests that are made through the standard public API to the underlying VIM APIs. The XAM Library is also responsible for loading the appropriate VIM that is needed to access a particular XAM Storage System. Thus, a particular XSystem is defined by the XAM Library-driven coupling of the VIM, the XAM Storage System components, and whatever configuration is done by storage system administrators.

The XAM Library decides how to dispatch the various field methods, based on the type of object referred to by the xam_handle_t. XSystem or XSet references are dispatched to the appropriate VIM; other references are handled by the XAM Library, itself, without referring to a VIM. This same pattern is followed for creating XIterators and for handling generation of error tokens from xam_status. Some field methods are only appropriate for XSets. The XAM Library is also responsible for handling these cases and not dispatching those requests that have semantic errors.

As noted above, the XAM Library is also responsible for locating and loading the VIM when a connect request is made. The XAM Library will preprocess the XRI to determine if the vimname is specified in the XRI. If it is specified, the XAM Library will load the specified VIM (or return an error if that VIM cannot be found or cannot be loaded). If a vimname is not provided, the XAM Library will search for an appropriate VIM and load the first VIM that meets the requirements of the XRI.

6.2 Methods

6.2.1 Error token generation

The XAM Library is directly responsible for token generation requests for all standard xam_status (i.e., it shall not invoke VIM methods). The VIM method defined in this section shall only be invoked when the application provides an XSystem handle, and the xam_status is vendor specific.

6.2.1.1 VIM_XSystem_GetErrorToken

Syntax prototype:

```
Xam_boolean
VIM_XSystem_GetErrorToken (const xsystem_handle inHandle,
                           const xam_status inStatus,
                           xam_string* const outToken);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference.
- inStatus is a valid xam_status.
- outToken is a reference to valid storage for a xam_string. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid xam_status.
- The third argument is NULL.

Description:

This method will generate an error token from the xam_status. This method is only responsible for generating tokens from xam_status that are vendor specific.

This method does not require the XSystem to be authenticated. It will also work on an XSystem that is in a corrupted or aborted state. It returns TRUE on success and FALSE on failure.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.2 Field iteration

The XAM Library is directly responsible for creating an XIterator when the specified xam_handle_t refers to the XAM_HANDLE (i.e., it shall not invoke VIM methods). The VIM methods defined in this section as creating field iterators shall only be invoked when the application provides an XSystem handle or an XSet handle. The other methods in this section are called from the matching public API call (as defined by the method name without the "VIM_" prefix).

6.2.2.1 VIM_XSystem_OpenFieldIterator

xam_status

```
VIM_XSystem_OpenFieldIterator (const xsystem_handle inHandle,
                               const xam_string inPattern,
                               xiterator_handle* const outIterator);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object contains the fields to be enumerated.
- inPattern is a valid xam_string containing a valid, NULL-terminated, UTF-8 byte sequence. The pattern in this xam_string will be used to filter the fields which will be enumerated. Those fields that do not belong with the specified pattern will not be included in the enumeration. The pattern is very simple; the byte sequence is treated as an explicit prefix. If the beginning of a field name does not match the exact bit sequence of the specified pattern, it will be filtered out of the results.
- outIterator is a reference to valid storage for an xiterator_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid prefix (invalid UTF-8).
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method acts as a factory interface, creating an XIterator from an XSystem. This iterator is used to discover the field names of fields on the XSystem. Only those fields whose names begin with the distinct bit sequence as specified in the pattern will be included in the enumeration.

Resources associated with the XIterator must be explicitly released. Once the resources are released, the XIterator will no longer be valid.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.2.2 VIM_XSet_OpenFieldIterator

Syntax prototype:

```
xam_status
VIM_XSet_OpenFieldIterator (const xset_handle inHandle,
                            const xam_string inPattern,
                            xiterator_handle* const outIterator);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the fields to be enumerated.
- inPattern is a valid xam_string containing a valid, NULL terminated UTF-8 byte sequence. The pattern in this xam_string will be used to filter the fields which will be enumerated – those fields that do not being with the specified pattern will not be included in the enumeration. The pattern is very simple – the byte sequence is treated as an explicit prefix, if the beginning of a field name does not match the exact bit sequence of the specified pattern it will be filtered out of the results.
- outIterator is a reference to valid storage for an xiterator_handle. The value that is passed in is not used and is overwritten with the result

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid prefix (invalid UTF-8).

- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method acts as a factory interface, creating an XIterator from an XSet. This iterator is used to discover the field names of fields on the XSet. Only those fields whose names begin with the distinct bit sequence as specified in the pattern will be included in the enumeration.

Resources associated with the XIterator must be explicitly released. Once the resources are released, the XIterator will no longer be valid.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.2.3 VIM_XIterator_Next

Syntax prototype:

```
xam_status
VIM_XIterator_Next (const xiterator_handle inHandle,
                    xam_string* const outName);
```

Parameters:

- inHandle is a valid xiterator_handle.
- outName is a reference to valid storage for a xam_string. The result is the name of the field following the current cursor (e.g., the field name of the field at the current cursor/position in the iteration). The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xiterator_handle.
- The second argument is NULL.
- Undefined errors will occur, if the resources associated with the XIterator have already been released.

Description:

This method copies the field name of the field at the current cursor of the iteration into the provided storage. The cursor is then advanced to the next field. On reading past the last field, an empty string will be returned.

Note: This method will only be invoked if the XAM Library cannot handle the request (i.e., when the XIterator was created against the XAM Library).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.2.4 VIM_XIterator_HasNext

Syntax prototype:

```
xam_status
VIM_XIterator_HasNext (const xiterator_handle inHandle,
                       xam_boolean* const outHasNext);
```

Parameters:

- inHandle is a valid xiterator_handle.
- outHasNext is a reference to valid storage for a xam_boolean. It is set to TRUE if there are more fields following the current cursor (e.g., after the field at the current cursor/position in the iteration). The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xiterator_handle.
- The second argument is NULL.
- Undefined errors will occur if the resources associated with the XIterator have already been released.

Description:

This method indicates of there are fields following the field at the current cursor of the iteration into the provided storage.

Note: This method will only be invoked if the XAM library cannot handle the request (i.e. when the XIterator was created against the XAM library).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.2.5 VIM_XIterator_Close

Syntax prototype:

```
xam_status
VIM_XIterator_Close (xiterator_handle inHandle);
```

Parameters:

- inHandle is a valid xiterator_handle.

Error conditions:

- The first argument is not a valid xiterator_handle.
- Undefined errors will occur, if the resources associated with the XIterator have already been released.
- The iterator is not an XSet or XSystem field iterator.

Description:

This method releases the resources associated with an open XIterator. After this method is called, the XIterator may no longer be used.

Note: This method will only be invoked if the XAM Library cannot handle the request (i.e., when the XIterator was created against the XAM Library).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3 Field manipulation

The XAM Library is directly responsible for manipulation of fields when the specified xam_handle_t refers to the XAM_HANDLE (i.e., it shall not invoke VIM methods to manipulate fields that are on the XAM Library object). The VIM methods defined in this section shall only be invoked when the application provides an XSystem handle or an XSet handle (i.e., when the fields reside on an XSystem or an XSet, respectively).

6.2.3.1 XSystem generic field methods

6.2.3.1.1 VIM_XSystem_ContainsField

Syntax prototype:

```
xam_status
VIM_XSystem_ContainsField (const xsystem_handle inHandle,
                           const xam_string inName,
                           xam_boolean* const outContained);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference.
- inName is a xam_string containing the name of the field.
- outContained is a reference to valid storage for a xam_boolean. It is set to TRUE if the field is contained in the XSystem. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will set the provided Boolean to TRUE if the field is contained in the XSystem. Otherwise, it will be set to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.1.2 VIM_XSystem_GetFieldType

Syntax prototype:

```
xam_status
VIM_XSystem_GetFieldType (const xsystem_handle inHandle,
                          const xam_string inName,
                          xam_string* const outType);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outType is a reference to valid storage for a xam_string. The result is the MIME type of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will copy the MIME type of the named field into the provided xam_string.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.1.3 VIM_XSystem_GetFieldLength

Syntax prototype:

```
xam_status
VIM_XSystem_GetFieldLength (const xsystem_handle inHandle,
                            const xam_string inName,
                            xam_int* const outLength);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outLength is a reference to valid storage for a xam_int. The result is the number of bytes of the value of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.

- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will copy the length of the named field into the provided xam_int.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.1.4 VIM_XSystem_GetFieldReadOnly

Syntax prototype:

```
xam_status
VIM_XSystem_GetFieldReadOnly (const xsystem_handle inHandle,
                              const xam_string inName,
                              xam_boolean* const outReadOnly);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outReadOnly is a reference to valid storage for a xam_boolean. The result is TRUE, if the readonly attribute of the named field is TRUE, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will set the xam_boolean value to TRUE, if the readonly attribute of the named field is TRUE, or to FALSE otherwise.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.1.5 VIM_XSystem_DeleteField

Syntax prototype:

```
xam_status
VIM_XSystem_DeleteField (const xsystem_handle inHandle,
                         const xam_string inName);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object contains the named field.
- inName is a xam_string containing the name of the field to delete.

Error conditions:

- The first argument is not a valid xystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will remove a field from the XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2 XSet generic field methods

6.2.3.2.1 VIM_XSet_ContainsField

Syntax prototype:

```
xam_status
VIM_XSet_ContainsField (const xset_handle inHandle,
                        const xam_string inName,
                        xam_boolean* const outContained);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference.
- inName is a xam_string containing the name of the field.
- outContained is a reference to valid storage for a xam_boolean. It is set to TRUE, if the field is contained in the XSet. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The third argument is NULL.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will set the provided Boolean to TRUE if the field is contained in the XSet. Otherwise it will be set to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.2 VIM_XSet_SetFieldAsBinding

Syntax prototype:

```
xam_status
VIM_XSet_SetFieldAsBinding (const xset_handle inHandle,
                            const xam_string inName);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t does not contain an XSet.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet that is in a corrupt state.
- The xam_handle_t contains an XSet that is in an abandoned state.

Description:

This method will set the binding attribute of a field to TRUE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.3 VIM_XSet_SetFieldAsNonbinding

Syntax prototype:

```
xam_status
VIM_XSet_SetFieldAsNonbinding (const xset_handle inHandle,
                               const xam_string inName);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t does not contain an XSet.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet that is in a corrupt state.
- The xam_handle_t contains an XSet that is in an abandoned state.

Description:

This method will set the binding attribute of a field to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.4 VIM_XSet_GetFieldType

Syntax prototype:

```
xam_status
VIM_XSet_GetFieldType (const xset_handle inHandle,
                        const xam_string inName,
                        xam_string* const outType);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outType is a reference to valid storage for a xam_string. The result is the MIME type of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.

- The third argument is NULL.
- The xset_handle contains an XSet and the XSet has an open import or export stream.
- The xset_handle contains an XSet that is in a corrupt state.
- The xset_handle contains an XSet that is in an aborted state.

Description:

This method will copy the MIME type of the named field into the provided xam_string.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.5 VIM_XSet_GetFieldLength

Syntax prototype:

```
xam_status
VIM_XSet_GetFieldLength (const xset_handle inHandle,
                         const xam_string inName,
                         xam_int* const outLength);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outLength is a reference to valid storage for a xam_int. The result is the number of bytes of the value of the named field in the object. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xset_handle contains an XSet and the XSet has an open import or export stream.
- The xset_handle contains an XSet that is in a corrupt state.
- The xset_handle contains an XSet that is in an aborted state.

Description:

This method will copy the length of the named field into the provided xam_int.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.6 VIM_XSet_GetFieldBinding

Syntax prototype:

```
xam_status
VIM_XSet_GetFieldBinding (const xset_handle inHandle,
                          const xam_string inName,
                          xam_boolean* const outBinding);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outBinding is a reference to valid storage for a xam_boolean. The result is TRUE if the binding attribute of the named field is TRUE or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xset_handle contains an XSet and the XSet has an open import or export stream.
- The xset_handle contains an XSet that is in a corrupt state.
- The xset_handle contains an XSet that is in an aborted state.

Description:

This method will set the xam_boolean value to TRUE, if the binding attribute of the named field is TRUE, or to FALSE otherwise.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.7 VIM_XSet_GetFieldReadOnly

Syntax prototype:

```
xam_status
VIM_XSet_GetFieldReadOnly (const xset_handle inHandle,
                           const xam_string inName,
                           xam_boolean* const outReadOnly);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to manipulate.
- outReadOnly is a reference to valid storage for a xam_boolean. The result is TRUE, if the readonly attribute of the named field is TRUE, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The xset_handle contains an XSet and the XSet has an open import or export stream.
- The xset_handle contains an XSet that is in a corrupt state.
- The xset_handle contains an XSet that is in an aborted state.

Description:

This method will set the xam_boolean value to TRUE, if the readonly attribute of the named field is TRUE, or to FALSE otherwise.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.2.8 VIM_XSet_DeleteField

Syntax prototype:

```
xam_status
VIM_XSet_DeleteField (const xset_handle inHandle,
                      const xam_string inName);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object contains the named field.
- inName is a xam_string containing the name of the field to delete.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The xam_handle_t contains an XSet was opened in readonly mode.
- The xam_handle_t contains an XSet was opened in restricted mode and the second argument contains a name that refers to a binding field.
- The xset_handle contains an XSet and the XSet has an open import or export stream.
- The xset_handle contains an XSet that is in a corrupt state.
- The xset_handle contains an XSet that is in an aborted state.

Description:

This method will remove a field from the XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3 XSystem property field methods

6.2.3.3.1 VIM_XSystem_CreateBoolean

Syntax prototype:

```
xam_status
VIM_XSystem_CreateBoolean (const xsystem_handle inHandle,
                           const xam_string inName,
                           const xam_boolean inBinding,
                           const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_boolean containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.boolean” on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.2 VIM_XSystem_CreateInt

Syntax prototype:

```
xam_status
VIM_XSystem_CreateInt (const xsystem_handle inHandle,
                       const xam_string inName,
                       const xam_boolean inBinding,
                       const xam_int inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_int containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.int” on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.3 VIM_XSystem_CreateDouble

Syntax prototype:

```
xam_status
VIM_XSystem_CreateDouble (const xsystem_handle inHandle,
                          const xam_string inName,
                          const xam_boolean inBinding,
                          const xam_double inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_double containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.double” on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.4 VIM_XSystem_CreateXUID

Syntax prototype:

```
xam_status
VIM_XSystem_CreateXUID (const xsystem_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inBinding,
                        const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_xuid containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.xuid” on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.5 VIM_XSystem_CreateString

Syntax prototype:

```
xam_status
VIM_XSystem_CreateString (const xsystem_handle inHandle,
                          const xam_string inName,
                          const xam_boolean inBinding,
                          const xam_string inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_string containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to "application/vnd.snia.xam.string" on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.6 VIM_XSystem_CreateDatetime

Syntax prototype:

```
xam_status
VIM_XSystem_CreateDatetime (const xsystem_handle inHandle,
                             const xam_string inName,
                             const xam_boolean inBinding,
                             const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_datetime containing the value to be stored.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The third argument is TRUE.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.datetime” on the XSystem instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.7 VIM_XSystem_SetBoolean

Syntax prototype:

```
xam_status
VIM_XSystem_SetBoolean (const xsystem_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_boolean containing the new value to be stored.

Error conditions:

- The named field is not of type Boolean.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.boolean” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.8 VIM_XSystem_SetInt

Syntax prototype:

```
xam_status
VIM_XSystem_SetInt (const xsystem_handle inHandle,
                    const xam_string inName,
                    const xam_int inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_int containing the new value to be stored.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.int” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.9 VIM_XSystem_SetDouble

Syntax prototype:

```
xam_status
VIM_XSystem_SetDouble (const xsystem_handle inHandle,
                       const xam_string inName,
                       const xam_double inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_double containing the new value to be stored.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.double” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.10VIM_XSystem_SetXUID

Syntax prototype:

```
xam_status
VIM_XSystem_SetXUID (const xsystem_handle inHandle,
                     const xam_string inName,
                     const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_xuid containing the new value to be stored.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.xuid” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.11 VIM_XSystem_SetString

Syntax prototype:

```
xam_status
VIM_XSystem_SetString (const xsystem_handle inHandle,
                       const xam_string inName,
                       const xam_string inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_string containing the new value to be stored.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.string” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.12VIM_XSystem_SetDatetime

Syntax prototype:

```
xam_status
VIM_XSystem_SetDatetime (const xsystem_handle inHandle,
                         const xam_string inName,
                         const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_datetime containing the new value to be stored.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.datetime” on the XSystem instance. Its value will be set according to the user-provided parameter.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.13VIM_XSystem_GetBoolean

Syntax prototype:

```
xam_status
VIM_XSystem_GetBoolean (const xsystem_handle inHandle,
                        const xam_string inName,
                        xam_boolean* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_boolean. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type Boolean.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.boolean” on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.14VIM_XSystem_GetInt

Syntax prototype:

```
xam_status
VIM_XSystem_GetInt (const xsystem_handle inHandle,
                    const xam_string inName,
                    xam_int* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_int. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to "application/vnd.snia.xam.int" on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.15VIM_XSystem_GetDouble

Syntax prototype:

```
xam_status
VIM_XSystem_GetDouble (const xsystem_handle inHandle,
                       const xam_string inName,
                       xam_double* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_double. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.

- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.double” on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.16VIM_XSystem_GetXUID

Syntax prototype:

```
xam_status
VIM_XSystem_GetXUID (const xsystem_handle inHandle,
                     const xam_string inName,
                     xam_xuid* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_xuid. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.xuid” on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.17 VIM_XSystem_GetString

Syntax prototype:

```
xam_status
VIM_XSystem_GetString (const xsystem_handle inHandle,
                        const xam_string inName,
                        xam_string* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_string. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.string” on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.3.18 VIM_XSystem_GetDatetime

Syntax prototype:

```
xam_status
VIM_XSystem_GetDatetime (const xsystem_handle inHandle,
                         const xam_string inName,
                         xam_datetime* const outValue);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_datetime. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.datetime” on the XSystem instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4 XSet property field methods

6.2.3.4.1 VIM_XSet_CreateBoolean

Syntax prototype:

```
xam_status
VIM_XSet_CreateBoolean (const xset_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inBinding,
                        const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_boolean containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.boolean” on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.2 VIM_XSet_CreateInt

Syntax prototype:

```
xam_status
VIM_XSet_CreateInt (const xset_handle inHandle,
                    const xam_string inName,
                    const xam_boolean inBinding,
                    const xam_int inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_int containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to "application/vnd.snia.xam.int" on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.3 VIM_XSet_CreateDouble

Syntax prototype:

```
xam_status
VIM_XSet_CreateDouble (const xset_handle inHandle,
                       const xam_string inName,
                       const xam_boolean inBinding,
                       const xam_double inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_double containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.double” on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.4 VIM_XSet_CreateXUID

Syntax prototype:

```
xam_status
VIM_XSet_CreateXUID (const xset_handle inHandle,
                     const xam_string inName,
                     const xam_boolean inBinding,
                     const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_xuid containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.xuid” on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.5 VIM_XSet_CreateString

Syntax prototype:

```
xam_status
VIM_XSet_CreateString (const xset_handle inHandle,
                       const xam_string inName,
                       const xam_boolean inBinding,
                       const xam_string inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_string containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.string” on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.6 VIM_XSet_CreateDatetime

Syntax prototype:

```
xam_status
VIM_XSet_CreateDatetime (const xset_handle inHandle,
                         const xam_string inName,
                         const xam_boolean inBinding,
                         const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inValue is a xam_datetime containing the value to be stored.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is being created as binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with a type set to “application/vnd.snia.xam.datetime” on the XSet instance. Its name, value, and binding attributes will be set according to the user-provided parameters.

Note: If binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.7 VIM_XSet_SetBoolean

Syntax prototype:

```
xam_status
VIM_XSet_SetBoolean (const xset_handle inHandle,
                     const xam_string inName,
                     const xam_boolean inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_boolean containing the new value to be stored.

Error conditions:

- The named field is not of type Boolean.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.boolean” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.8 VIM_XSet_SetInt

Syntax prototype:

```
xam_status
VIM_XSet_SetInt (const xset_handle inHandle,
                 const xam_string inName,
                 const xam_int inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_int containing the new value to be stored.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.int” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.9 VIM_XSet_SetDouble

Syntax prototype:

```
xam_status
VIM_XSet_SetDouble (const xset_handle inHandle,
                    const xam_string inName,
                    const xam_double inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_double containing the new value to be stored.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.double” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.10 VIM_XSet_SetXUID

Syntax prototype:

```
xam_status
VIM_XSet_SetXUID (const xset_handle inHandle,
                  const xam_string inName,
                  const xam_xuid inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_xuid containing the new value to be stored.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.xuid” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.11 VIM_XSet_SetString

Syntax prototype:

```
xam_status
VIM_XSet_SetString (const xset_handle inHandle,
                    const xam_string inName,
                    const xam_string inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_string containing the new value to be stored.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.string” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.12VIM_XSet_SetDatetime

Syntax prototype:

```
xam_status
VIM_XSet_SetDatetime (const xset_handle inHandle,
                      const xam_string inName,
                      const xam_datetime inValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inValue is a xam_datetime containing the new value to be stored.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will change a property field with a type set to “application/vnd.snia.xam.datetime” on the XSet instance. Its value will be set according to the user-provided parameter.

Note: If the field is binding, a new XSet is created and a new XUID will be assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.13VIM_XSet_GetBoolean

Syntax prototype:

```
xam_status
VIM_XSet_GetBoolean (const xset_handle inHandle,
                     const xam_string inName,
                     xam_boolean* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_boolean. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type Boolean.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.boolean” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.14 VIM_XSet_GetInt

Syntax prototype:

```
xam_status
VIM_XSet_GetInt (const xset_handle inHandle,
                 const xam_string inName,
                 xam_int* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_int. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type int.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.int” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.15VIM_XSet_GetDouble

Syntax prototype:

```
xam_status
VIM_XSet_GetDouble (const xset_handle inHandle,
                     const xam_string inName,
                     xam_double* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_double. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type double.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.double” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.16 VIM_XSet_GetXUID

Syntax prototype:

```
xam_status
VIM_XSet_GetXUID (const xset_handle inHandle,
                  const xam_string inName,
                  xam_xuid* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_xuid. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type XUID.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.xuid” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.17 VIM_XSet_GetString

Syntax prototype:

```
xam_status
VIM_XSet_GetString (const xset_handle inHandle,
                    const xam_string inName,
                    xam_string* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_string. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type string.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.string” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.4.18VIM_XSet_GetDatetime

Syntax prototype:

```
xam_status
VIM_XSet_GetDatetime (const xset_handle inHandle,
                      const xam_string inName,
                      xam_datetime* const outValue);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- outValue is a reference to valid storage for a xam_datetime. The value of the named field is written into this value. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The named field is not of type datetime.
- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will get the value from a property field with a type set to “application/vnd.snia.xam.datetime” on the XSet instance.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5 XStream field methods

6.2.3.5.1 VIM_XSystem_CreateXStream

Syntax prototype:

```
xam_status
VIM_XSystem_CreateXStream (const xsystem_handle inHandle,
                           const xam_string inName,
                           const xam_boolean inBinding,
                           const xam_string inType,
                           xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inType is a xam_string that contains the MIME type of the field.
- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The fourth argument contains an empty string ("" is not a valid MIME type).
- The fifth argument contains a NULL.
- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create an XStream field with a type set to the user-defined MIME type on the XSystem instance. Its name, MIME type, and binding attributes will be set according to the user-provided parameters. The XStream field is opened in writeonly mode.

Note: The value is not set by the method. This method will create an XStream with a length of zero; other methods must be used to add data to this field.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.2 VIM_XSet_CreateXStream

Syntax prototype:

```
xam_status
VIM_XSet_CreateXStream (const xset_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inBinding,
                        const xam_string inType,
                        xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be created.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inType is a xam_string that contains the MIME type of the field.
- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field that is not legal for applications to create.
- The second argument contains a name of a field that is already in use.
- The fourth argument contains an empty string ("" is not a valid MIME type).
- The fifth argument contains a NULL.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the field is binding.
- The XSet was opened in restricted mode and is on hold.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of XStream fields allowed on this XSet has been reached.

Description:

This method will create an XStream field with a type set to the user-defined MIME type on the XSet instance. Its name, MIME type, and binding attributes will be set according to the user-provided parameters. The XStream field is opened in writeonly mode.

Note: The value is not set by the method. This method will create an XStream with a length of zero; other methods must be used to add data to this field.

Note: This method may fail with an error, if the maximum number of fields that are supported on an XSet is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate the *.xsystem.limits.maxFieldsPerXSet* field on the XSystem instance. For more information on this topic, please consult the [XAM-ARCH].

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.3 VIM_XSystem_OpenXStream

Syntax prototype:

```
xam_status
VIM_XSystem_OpenXStream (const xsystem_handle inHandle,
                         const xam_string inName,
                         const xam_string inMode,
                         xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xsystem_handle containing an XSystem reference.
- inName is a xam_string containing the name of the field to be opened.
- inMode is a string indicating the mode to open the XStream in:
  - — readonly: open for reading. Write methods will fail on the XStream instance.
  - — writeonly: open for writing. Read and seek methods will fail on the XStream instance.
- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument contains anything other than a writeonly or a readonly.
- The fourth argument is NULL.

- The XSystem is in a corrupt state.
- The XSystem is in an abandoned state.

Description:

This method will create an open XStream in either readonly or writeonly mode, based on the mode argument.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.4 VIM_XSet_OpenXStream

Syntax prototype:

```
xam_status
VIM_XSet_OpenXStream (const xset_handle inHandle,
                      const xam_string inName,
                      const xam_string inMode,
                      xstream_handle* const outXStream);
```

Parameters:

- inHandle is a valid xset_handle containing an XSet reference. This object will contain the new field.
- inName is a xam_string containing the name of the field.
- inMode is a string indicating the mode to open the XStream in:
  - — readonly: open for reading. Write methods will fail on the XStream instance.
  - — writeonly: open for writing. Read and seek methods will fail on the XStream instance.
- outXStream is a reference to valid storage for an xstream_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument contains anything other than writeonly or a readonly.
- The fourth argument is NULL.
- The XSet was opened in readonly mode and the XStream open mode is writeonly.

- The XSet was opened in restricted mode, the field is binding, and the XStream open mode is writeonly.
- The XSet is on hold and the XStream open mode is writeonly.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create an open XStream in either readonly or writeonly mode, based on the mode argument.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.5 VIM_XStream_Read

Syntax prototype:

```
xam_status
VIM_XStream_Read (const xstream_handle inHandle,
                  char* const ioBuffer,
                  const xam_int inBufferLength,
                  xam_int* const outBytesRead);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in read mode.
- ioBuffer is a byte array to read the data into.
- inBufferLength is a xam_int set to the number of bytes in the buffer.
- outBytesRead is a reference to valid storage for a xam_int. On return, this value will contain the actual number of bytes read. This value will be less than or equal to the inBufferLength. When there is no more data to be read, a value of -1 will be set. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is not an XStream that was opened in readonly mode.
- The second argument is NULL.
- The buffer length is less than or equal to zero.

Note: If the inBufferLength is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method transfers data from the storage system into the target buffer, up to the number of bytes requested.

Concurrency requirements:

This method is thread safe.

Blocking:

This method does not block until data is completely read, but will indicate the amount of data that was read in each call. Subsequent calls may be needed to read the remainder of the data.

6.2.3.5.6 VIM_XStream_Write

Syntax prototype:

```
xam_status
VIM_XStream_Write (const xstream_handle inHandle,
                   const char* const inBuffer,
                   const xam_int inByteCount,
                   xam_int* const outByteWritten);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in writeonly mode.
- inBuffer is a byte array containing the data to be written.
- inByteCount is a xam_int set to the number of bytes in the buffer to be written.
- outBytesWritten is a reference to valid storage for a xam_int. On return, this value will contain the actual number of bytes written. This method will be less than or equal to the inByteCount. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is not an XStream that was opened in writeonly mode.
- The second argument is NULL.
- The maximum length (in bytes) of an XStream is exceeded.

Note: If the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method transfers data from the source buffer to the XAM Storage System, up to the number of bytes requested.

Note: This method may fail with an error, if the maximum number of bytes supported in an XStream is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate the *.xsystem.limits.maxSizeOfXStream* field on the XSystem instance. For more information on this topic, please consult the [XAM-ARCH].

Concurrency requirements:

This method is thread safe.

Blocking:

This method does not block until all the data in the buffer is completely written, but it will indicate the amount of data that was written in each call. Subsequent calls may be needed to write all of the data.

6.2.3.5.7 VIM_XStream_Seek

Syntax prototype:

```
xam_status
VIM_XStream_Seek (const xstream_handle inHandle,
                  const xam_int inOffset,
                  const xam_int inWhence);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in read mode.
- inOffset is a xam_int containing the number of bytes to change the position by.
- inWhence is a xam_int containing a 0, 1, or 2 (indicating where the offset should be measured from). These are defined as follows:
  - 0: The offset is measured from the start of the XStream.
  - 1: The offset is measured from the current position in the XStream.
  - 2: The offset is measured from the end of the XStream

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is not an XStream that was opened in readonly mode.
- The second and third arguments would result in a new position before the first byte in the XStream, or past the final byte in the XStream.
- The third argument contains a value other than 0, 1, or 2.

Description:

This method sets the position indicator for the XStream. The new position, measured in bytes, is obtained by adding inOffset bytes to the position specified by inWhence. If inWhence is set to 0, 1, or 2, then the offset is relative to the start of the XStream, the current position, or end-of-data, respectively.

Note: This method can only be used for XStreams opened for read. In addition, this method cannot be used to create sparse files. It is an error to seek past the end of the data in the XStream, as indicated by the field attribute ‘length’.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.8 VIM_XStream_Tell

Syntax prototype:

```
xam_status
VIM_XStream_Tell (const xstream_handle inHandle,
                  xam_int* const outPosition);
```

Parameters:

- inHandle is an xstream_handle.
- outPosition is a xam_int containing the position in the XStream.

Error conditions:

- The first argument is not a valid xstream_handle.
- The second argument is NULL.

Description:

This method gets the current value of the XStream position indicator.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.9 VIM_XStream_Abandon

Syntax prototype:

```
xam_status
VIM_XStream_Abandon (const xstream_handle inHandle);
```

Parameters:

- inHandle is an xstream_handle.

Error conditions:

- The first argument is not a valid xstream_handle.

CAUTION:	If the XStream has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XStream in its normal state will generate an error, when an application attempts to close it, if there are open asynchronous operations being performed on it. Making this call will change the state of the XStream and allow it to be closed, without regard for any open asynchronous operations.

Note: The XStream will no longer be usable after this call is made, and the only call that will succeed is XStream.close.

CAUTION:	This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open asynchronous operations and close the asynchronous operations properly as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.3.5.10VIM_XStream_Close

Syntax prototype:

```
xam_status
VIM_XStream_Close (xstream_handle inHandle);
```

Parameters:

- inHandle is an xstream_handle.

Error conditions:

- The first argument is not a valid xstream_handle.

CAUTION:	Closing an already closed XStream can produce undefined results, including data loss and data corruption)

Description:

This method closes a previously opened XStream. Any resources that were allocated can be released at this point.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.4 Connection administration for a XAM Storage System

When XAMLibrary_Connect is called by the application, the XAM Library is responsible for creating the XSystem instance, updating the fields on the new XSystem instance, and then preprocessing the XRI to determine if the vimname is specified in the XRI. If a vimname is not provided, the XAM Library will search for an appropriate VIM and load the first VIM that meets the requirements of the XRI. After the VIM is loaded, the XRI is passed to the VIM using the connect method defined in this section. The other methods in this section are called from the matching API call (as defined by the method name without the "VIM_" prefix).

When the application creates an XSystem instance (using an XRI and the XAMLibrary.connect method) the XAMLibrary shall load and initialize the VIM. Loading and initializing the VIM shall not require any special methods to be invoked by the calling application; this is done automatically as a part of the connect. The transfer of information from the XAM Library to the VIM is mediated by the XSystem instance. When constructed, a field shall be created on the XSystem instance. This field shall be named *.xsystem.initializing* with a value of TRUE and with readonly also being TRUE. Then, all fields on the XAM Library shall be copied onto the new XSystem instance. Finally, the *.xsystem.initializing* field will be removed. The VIM shall take this information and process it accordingly. Finally, the unauthenticated XSystem instance shall be returned to the application.

6.2.4.1 VIM_CreateXSystem

Syntax prototype:

```
xam_status
VIM_CreateXSystem (xsystem_handle* const outHandle);
```

Parameters:

- outHandle is a reference to valid storage for an xsystem_handle. On return, this value will contain the XSystem handle that was created. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is NULL.

Description:

This method will create a new XSystem instance, containing a single xam_boolean field. This field shall be named *.xsystem.initializing* with a value of TRUE and with readonly also being TRUE. When this field is present and set to TRUE, the XSystem instance can have fields created on it only. Other methods (with the exception of Connect, Abandon, and Close) will generate non-fatal errors. This method is invoked by the XAMLIbrary_Connect method, and the resultant XSystem instance should not be exposed through the public interfaces, until the fields on the XAM Library are copied to it and the private connect method is called.

Note: The XSystem instance is not usable to applications until it has been connected to a XAM Storage System and has been authenticated.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.4.2 VIM_XSystem_Connect

Syntax prototype:

```
xam_status
VIM_XSystem_Connect (const xsystem_handle inHandle,
                     const xam_string inXRI);
```

Parameters:

- inHandle is an xsystem_handle.
- inXRI is a xam_string. It contains the XSystem's Uniform Resource Identifier. The format of the XRI is listed below:

  ```
  snia-xam://[vimname!]xsystemname[?param=value[{&param=value}]]
  ```

  The vimname is a string that describes which VIM to use, and if it is not specified, the XAM system will choose a VIM to use. A vimname is not allowed to contain a '!' character. The xsystemname is vendor specific; it may be an IP address or some other id. It may not contain '/', '?', or '!' characters. Finally, param'='value pairs can be specified. The full BNF of this format can be found in the XAM Architecture Specification [XAM-ARCH].

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is not a valid XRI.
- The underlying XAM Storage System or its infrastructure (e.g., a damaged cable for IP attached storage) has a problem.

Description:

This method takes an XSystem instance that contains *.xsystem.initializing* and connects it to a specific XAM Storage System. When called, it removes *.xsystem.initializing* from the XSystem instance, and then evaluates the XSystem's Uniform Resource Identifier (XRI) string. It is expected that the XRI will be specified by the local storage system administrators, and applications should strive to make this easily configured at run time.

Note: The XSystem instance is not usable until it has been authenticated.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.4.3 VIM_XSystem_Authenticate

Syntax prototype:

```
xam_status
VIM_XSystem_Authenticate (const xsystem_handle inHandle,
                          const char* const inBuffer,
                          const xam_int inByteCount,
                          xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xsystem_handle.
- inBuffer: Data that is being passed to the authentication mechanism is passed in this array of bytes.
- inByteCount: The number of significant bytes in the passed-in buffer.
- outXStream is a reference to valid storage for an xstream_handle. On return, this will contain the XStream handle that was created, which contains the system's response to the authentication information. The value that is passed in is not used and is overwritten with the result.

Note: The outXStream must be closed when the application has finished its authentication processing.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The fourth argument is NULL.
- Authentication fails.

Note: If the XSystem has been closed, or if the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method allows an application to authenticate an XSystem. It provides a generic interface to exchange data as part of the authentication process. The application should check for XSystem instance properties with the prefix of *.xsystem.auth.SASLmechanism.list.* to determine which patterns of authentication (mechanisms) are available for use. After a pattern is selected, the appropriate sequence of data exchanges should be made (using this call), in order to authenticate. A failed authentication will make the XSystem unusable; applications cannot repeat failed authentications using the same XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.4.4 VIM_XSystem_Close

Syntax prototype:

```
xam_status
VIM_XSystem_Close (const xsystem_handle inHandle);
```

Parameters:

- inHandle is an xsystem_handle.

Error conditions:

- The first argument is not a valid xsystem_handle.
- There are open XSets or XStreams.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is called to release any resources associated with an XSystem. After calling this method, the closed XSystem should not be used.

Note: This call will fail if there are any open XSets associated with this XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.4.5 VIM_XSystem_Abandon

Syntax prototype:

```
xam_status
VIM_XSystem_Abandon (const xsystem_handle inHandle);
```

Parameters:

- inHandle is an xsystem_handle.

Error conditions:

- The first argument is not a valid xsystem_handle.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XSystem, in its normal state, will generate an error, when an application attempts to close it, if it has open XSets in it. Making this call will change the state of the XSystem and allow it to be closed without regard for any open XSets.

Note: The XSystem will no longer be usable after this call is made, and the only call that will succeed is XSystem.close.

CAUTION: This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open XSets and close the XSets properly as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.5 XSet instance creation

When applications create an XSet, the XAM Library must return a non-zero error status when the specified xam_handle_t refers to the XAM_HANDLE or an XSet handle, and it shall not invoke VIM methods in these cases. The VIM methods defined in this section shall only be called from the matching API call (as defined by the method name without the "VIM_" prefix), when the application provides an XSystem handle.

6.2.5.1 VIM_XSystem_CreateXSet

Syntax prototype:

```
xam_status
VIM_XSystem_CreateXSet (const xsystem_handle inHandle,
                        const xam_string inMode,
                        xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inMode is a string indicating the mode to open the XSet in:
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- outXSet is a reference to valid storage for an xset_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is NULL.
- The second argument is not restricted or unrestricted.
- The third argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will create a new, empty XSet instance associated with the XSystem. This XSet will not exist on the XSystem unless that XSet instance is committed.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.5.2 VIM_XSystem_OpenXSet

Syntax prototype:

```
xam_status
VIM_XSystem_OpenXSet (const xsystem_handle inHandle,
                      const xam_xuid inXUID,
                      const xam_string inMode,
                      xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be opened.
- inMode is a string indicating the mode to open the XSet in:
  - — readonly: open for reading. Adding, deleting, or modifying fields is not allowed. Commit of the XSet instance will fail.
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or on changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).

- outXSet is a reference to valid storage for a xset_handle. On return, this value will contain the XSet handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not readonly, restricted, or unrestricted.
- The XSet is on hold, and the mode is not readonly.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an XSet in the XSystem, returning a handle to an XSet instance associated with the XSystem. This XSet will not exist on the XSystem, unless that XSet instance is committed.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.5.3 VIM_XSystem_CopyXSet

Syntax prototype:

```
xam_status
VIM_XSystem_CopyXSet (const xsystem_handle inHandle,
                      const xam_xuid inXUID,
                      const xam_string inMode,
                      xset_handle* const outXSet);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be opened.
- inMode is a string indicating the mode to open the XSet in:

   - restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
   - unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or on changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- outXSet is a reference to valid storage for a xset_handle. On return, this value will contain the XSet handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not restricted or unrestricted.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will create a copy of an XSet in the XSystem, returning a handle to an XSet instance associated with the XSystem. This XSet will not exist on the XSystem unless that XSet instance is committed.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6 XSet administration

When applications invoke XSet management methods, the XAM Library must return a non-zero error status when the specified xam_handle_t refers to the XAM_HANDLE or an XSet handle, and it shall not invoke VIM methods in these cases. The VIM methods defined in this section shall only be called from the matching API call (as defined by the method name without the “VIM_” prefix) when the application provides an XSystem handle.

6.2.6.1 VIM_XSystem_IsXSetRetained

Syntax prototype:

```
xam_status
VIM_XSystem_IsXSetRetained (const xsystem_handle inHandle,
                            const xam_xuid inXUID,
                            xam_boolean* const outIsRetained);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- outIsRetained is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE, if the XSet is accessible, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will evaluate all retention criteria that exists on the specified XSet and shall return TRUE if there exists retention criterion which would prevent XSet deletion. The method returns FALSE if the retention criteria are not sufficient to describe a complete retention, if the retention is not enabled, or if the retention criteria are valid but the retention period has passed.

This method does not evaluate the “on-hold” status.

A non-fatal error will be returned if the specified XUID is improperly formatted, does not exist in the XSystem, or if the caller is not authorized to read the XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6.2 VIM_XSystem_DeleteXSet

Syntax prototype:

```
xam_status
```

```
VIM_XSystem_DeleteXSet (const xsystem_handle inHandle,
                        const xam_xuid inXUID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be deleted.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The XSystem does not have authorization to delete an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will delete an XSet from the XSystem.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6.3 VIM_XSystem_HoldXSet

Syntax prototype:

```
xam_status
VIM_XSystem_HoldXSet (const xsystem_handle inHandle,
                      const xam_xuid inXUID,
                      const xam_string inHoldID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be held.
- inHoldID is a xam_string that contains the ID to be associated with the hold.

Error conditions:

- The first argument is not a valid xsystem_handle.

- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The third argument contains a hold id that is already in use for this XSet.
- The XSystem does not have authorization to hold an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will place an XSet on hold. A held XSet cannot be changed in any way (e.g., the XSet can only be opened in readonly mode, and commits of a held XSet will fail).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6.4 VIM_XSystem_ReleaseXSet

Syntax prototype:

```
xam_status
VIM_XSystem_ReleaseXSet (const xsystem_handle inHandle,
                         const xam_xuid inXUID,
                         const xam_string inHoldID);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be held.
- inHoldID is a xam_string that contains the ID associated with the hold.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The second argument contains a XUID of an XSet that does not exist (or is not accessible) in the XSystem.
- The third argument contains a hold id that is not in use for this XSet.
- The XSet is not being held at all.

- The XSystem does not have authorization to release a hold from an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will release a specific hold on an XSet (associated with the hold id).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6.5 VIM_XSystem_AccessXSet

Syntax prototype:

```
xam_status
VIM_XSystem_AccessXSet (const xsystem_handle inHandle,
                        const xam_xuid inXUID,
                        const xam_int inMode,
                        xam_boolean* const outIsAccessible);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- inMode: The value is the bitwise OR of the access 'permissions' to be checked (R_OK for read permission, WU_OK for write-user permission, WS_OK for write-system permission, D_OK for delete, H_OK for hold, RE_OK for retention event, J_OK for job and JC_OK for job commit). In addition, there are composite permissions W_OK (WU_OK|WS_OK), RW_OK (R_OK|W_OK) and ALL_OK (RW_OK|D_OK|H_OK|RE_OK|J_OK|JC_OK).
- outIsAccessible is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE, if the XSet is accessible, or FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument does not contain a valid mode.
- The fourth argument is NULL.
- The XSystem does not have authorization to query an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will check the accessibility of an XSet on the XSystem. It is not an error if the XSet does not exist on the XSystem. Such an XSet is noted as being inaccessible.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.6.6 VIM_XSystem_GetXSetAccessTime

Syntax prototype:

```
xam_status
VIM_XSystem_AccessXSet (const xsystem_handle inHandle,
                        const xam_xuid inXUID,
                        xam_datetime* const outAccessTime);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be checked.
- outAccessTime is a reference to valid storage for a xam_datetime. On return, this value will be set to the time at which the XSet was last opened or committed, whichever is the most recent. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The XSystem does not have authorization to evaluate the access time of an XSet.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will get the time at which the XSet was last opened or committed, whichever is the most recent.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7 XSet management administration

When applications invoke XSet policy management methods, the XAM Library must return a non-zero error status, when the specified xam_handle_t refers to the XAM_HANDLE or an XSystem, and it shall not invoke VIM methods in these cases. The VIM methods that are defined in this section shall only be invoked when the application provides an XSet handle.

6.2.7.1 Access policy

6.2.7.1.1 VIM_XSet_ApplyAccessPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyAccessPolicy (const xset_handle inHandle,
                            const xam_boolean inBinding,
                            const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with the name of *.xset.access.policy* and a type set to "application/vnd.snia.xam.string" on the object referenced by the passed-in xam_handle_t. Its value and binding attributes will be set according to the user-provided parameters. This field will be used by the XAM Storage System to determine the policies to use when accessing this XSet.

Note: If an access policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default access policy of the XSystem (i.e., the first string in *.xsystem.access.policy.list.<name>*).

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.1.2 VIM_XSet_ResetAccessFields

Syntax prototype:

```
xam_status
VIM_XSet_ResetAccessFields (const xset_handle inHandle);
```

Parameters:

- inHandle is a valid xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will remove all access fields from the XSet.

Note: If an access policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default access policy of the XSystem (i.e., the first string in *.xsystem.access.policy.list.<name>*).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.2 Base management policy

6.2.7.2.1 VIM_XSet_ApplyManagementPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyManagementPolicy (const xset_handle inHandle,
                                 const xam_boolean inBinding,
                                 const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will create a property field with the name of *.xset.management.policy* and a type set to “application/vnd.snia.xam.string” on the object referenced by the passed-in xam_handle_t. Its value and binding attributes will be set according to the user-provided parameters. This field will be used by the XAM Storage System to determine the default policies to use when managing this XSet.

Note: If the base management policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default management policy of the XSystem (i.e., *.xsystem.management.policy.default*).

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.2.2 VIM_XSet_ResetManagementFields

Syntax prototype:

```
xam_status
VIM_XSet_ResetManagementFields (const xset_handle inHandle);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

Description:

This method will remove all management fields from the XSet. Calling this method will result in a new XSet being created and a new XUID being assigned to this XSet at successful commit.

Note: If the base management policy has not been applied to an XSet at the time of the initial commit, then the property will be created and set as the default management policy of the XSystem (i.e., *.xsystem.management.policy.default*).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3 Retention

6.2.7.3.1 VIM_XSet_CreateRetention

Syntax prototype:

```
xam_status
VIM_XSet_CreateRetention (const xset_handle inHandle,
                          const xam_boolean inBinding,
                          const xam_string inRetentionID);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.

- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain a validly formatted retention identifier.
- The retention identifier already exists in the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The retention identifier is “base”
- The retention identifier is “event” and the binding input parameter is FALSE.

Description:

This method will create a scope for storing and evaluating retention criteria. It creates a field with a type of “application/vnd.snia.xam.string” and sets the value to the retention id. The field name is formed by appending the retention id to the following prefix: *.xset.retention.list.* Thus, the final format of the name is *.xset.retention.list.<retention id>*. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Creating a binding set of retention criteria will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.2 VIM_XSet_SetRetentionEnabledFlag

Syntax prototype:

```
xam_status
VIM_XSet_SetRetentionEnabledFlag (const xset_handle inHandle,
                                  const xam_string inRetentionID,
                                  const xam_boolean inBinding,
                                  const xam_boolean inEnabled);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inEnabled is a xam_boolean containing a flag indicating if event retention is enabled on this XSet or not. If the flag is set to TRUE, event retention is enabled; otherwise, it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- Enabled is being set to FALSE after it was set to TRUE.
- The retention identifier is “base”.

Description:

This method will enable or disable retention that is scoped by the specified retention id. This flag is stored in a field of type “application/vnd.snia.xam.boolean”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.enabled*); thus, the final format of the name is *.xset.retention.<retention id>.enabled*. If the field does not exist, it will be created; otherwise the value will be updated only if the value is changed from FALSE to TRUE. if the value is set to TRUE, it cannot be changed. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.3 VIM_XSet_ApplyRetentionEnabledPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyRetentionEnabledPolicy (const xset_handle inHandle,
                                      const xam_string inRetentionID,
                                      const xam_boolean inBinding,
                                      const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The applied policy has the effect of disabling retention for this retention ID after it was previously enabled.
- The retention identifier is “base”.

Description:

This method will enabled or disable retention that is scoped by the specified retention id. The policy name of the policy holding the enabled flag is stored in a field of type “application/vnd.snia.xam.string”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.enabled.policy*); thus, the final format of the name is *.xset.retention.<retention id>.enabled.policy*. If the field does not exist, it will be created; otherwise the value will be updated only if the value is changed from FALSE to TRUE. If the value is set to TRUE, it cannot be changed. It will have its binding attribute set according to the binding flag that is set by the application.

Note: If the *.xset.retention.<retention id>.enabled* field is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.4 VIM_XSet_SetRetentionDuration

Syntax prototype:

```
xam_status
VIM_XSet_SetRetentionDuration (const xset_handle inHandle,
                               const xam_string inRetentionID,
                               const xam_boolean inBinding,
                               const xam_int inDuration);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inDuration is a xam_int containing the amount of time (measured in milliseconds from the time of commit) to retain the XSet. Zero indicates no retention, while a negative one (-1) indicates infinite retention.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain a valid duration.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.

- The field already exists on the XSet, and the specified duration value is less than the existing duration value.
- The retention identifier is “base”.

Description:

This method will set the duration of retention that is scoped by the specified retention id. This flag is stored in a field of type “application/vnd.snia.xam.int”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.duration*); thus, the final format of the name is *.xset.retention.<retention id>.duration*. If the field does not exist, it will be created; otherwise the value will be updated only if the duration is increased. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.5 VIM_XSet_ApplyRetentionDurationPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyRetentionDurationPolicy (const xset_handle inHandle,
                                       const xset_string inRetentionID,
                                       const xam_boolean inBinding,
                                       const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The fourth argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.

- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- The applied policy has the effect of decreasing the duration for this retention ID.
- The retention identifier is “base”.

Description:

This method will set the duration of retention that is scoped by the specified retention id. This policy name is stored in a field of type “application/vnd.snia.xam.string”. The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.duration.policy*); thus, the final format of the name is *.xset.retention.<retention id>.duration.policy*. If the field does not exist, it will be created; otherwise the value will be updated only if the duration is increased. It will have its binding attribute set according to the binding flag that is set by the application.

Note: If *.xset.retention.<retention id>.duration* is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.6 VIM_XSet_SetRetentionStarttime

Syntax prototype:

```
xam_status
VIM_XSet_SetRetentionStarttime (const xset_handle inHandle,
                                const xam_string inRetentionID,
                                const xam_boolean inBinding);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inRetentionID is a xam_string containing the retention identifier of the retention being enabled or disabled.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention that is scoped by the retention identifier has not been created on the XSet.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The maximum number of fields allowed on this XSet has been reached.
- This method has already been used on an XSet.
- The retention identifier is "base".

Description:

This method will set the start time of retention that is scoped by the specified retention id. The current time of the XSystem is stored in a field of type "application/vnd.snia.xam.datetime". The name of the field is formed by inserting the retention id between a prefix (*.xset.retention.*) and a suffix (*.starttime*); thus, the final format of the name is *.xset.retention.<retention id>.starttime*. If the field does not exist, it will be created; otherwise, an error will be generated, as it is not allowed to change the start time once set. It will have its binding attribute set according to the binding flag that is set by the application.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.7 VIM_XSet_SetBaseRetention

Syntax prototype:

```
xam_status
VIM_XSet_SetBaseRetention (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_int inDuration);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.

- inDuration is a xam_int containing the amount of time (measured in milliseconds from the time of commit) to retain the XSet. Zero indicates no retention, while a negative one (-1) indicates infinite retention.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain a valid duration.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.
- The field already exists on the XSet, and the specified duration value is less than the existing duration value.

Description:

If this XSet does not already contain the field *.xset.retention.list.base*, this method will create the field with a type of "application/vnd.snia.xam.string" and set the value to "base". It will also create the "application/vnd.snia.xam.boolean" field *.xset.retention.base.enabled* and set the value to TRUE. The duration will be stored in a field named *.xset.retention.base.duration*. This field is of type "application/vnd.snia.xam.int". If the field already exists, its value will be changed to match the passed in duration only if the duration of the retention is not reduced; the method will generate an error if the duration is reduced. If the field does not already exist, it will be created with the specified duration as the value. The *.xset.retention.base.duration* field will have its binding attribute set according to the binding flag that is set by the application. The *.xset.retention.list.base* is always a binding field.

These fields will be used by the XAM Storage System to determine the base retention duration to use when managing this XSet.

Note: Changing *.xset.retention.base.duration* from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Note: When an XSet instance containing the field *.xset.retention.list.base* is first committed, the field *.xset.retention.base.starttime* will be created as a binding field and have its value set to *.xset.time.xuid.*

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.3.8 VIM_XSet_ApplyBaseRetentionPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyBaseRetentionPolicy (const xset_handle inHandle,
                                   const xam_boolean inBinding,
                                   const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet that was opened in readonly mode.
- The XSet that was opened in restricted mode and the field being created is a binding field.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not already contain the field *.xset.retention.list.base*, this method will create the field with a type of "application/vnd.snia.xam.string" and set the value to "base". It will also create the "application/ vnd.snia.xam.boolean" field *.xset.retention.base.enabled* and set the value to TRUE. The duration policy will be stored in a field named *.xset.retention.base.duration.policy*. This field is of type "application/ vnd.snia.xam.string". If the field already exists, its value will be changed to match the passed-in policy, only if the policy would not reduce the duration of the retention; the method will generate an error if the policy reduces the duration. If the field does not already exist, it will be created with the specified policy name as the value. These fields will have their binding attribute set according to the binding flag that is set by the application.

These fields will be used by the XAM Storage System to determine the base retention duration to use when managing this XSet.

Note: If the *.xset.retention.base.duration* field is also present on the XSet, it will be used by the XAM Storage System in preference to this policy field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Note: When an XSet instance containing the field *.xset.retention.list.base* is first committed, the field *.xset.retention.base.starttime* will be created and have its value set to *.xset.time.xuid*.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.4 AutoDelete

6.2.7.4.1 VIM_XSet_ApplyAutoDeletePolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyAutoDeletePolicy (const xset_handle inHandle,
                                const xam_boolean inBinding,
                                const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have an auto delete policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.deletion.autodelete.policy* and a type set to “application/vnd.snia.xam.string”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically deleted when retention expires.

Note: If *.xset.deletion.autodelete* is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.4.2 VIM_XSet_SetAutoDelete

Syntax prototype:

```
xam_status
VIM_XSet_SetAutoDelete (const xset_handle inHandle,
                        const xam_boolean inBinding,
                        const xam_boolean inAutoDelete);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inAutoDelete is a xam_boolean containing a flag indicating if autodelete is enabled on this XSet or not. If the flag is set to TRUE, autodelete is enabled; otherwise, it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have auto delete set on it, this method will create a property field on the specified XSet with the name of *.xset.deletion.autodelete* and a type set to “application/vnd.snia.xam.boolean”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically deleted when retention expires.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.5 Shred

6.2.7.5.1 VIM_XSet_ApplyShredPolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyShredPolicy (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have an auto shred policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.deletion.shred.policy* and a type set to “application/ vnd.snia.xam.string”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically shredded when it's deleted.

Note: If *.xset.deletion.shred* is also present on the XSet, it will be used by the XAM Storage System in preference to this field.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.5.2 VIM_XSet_SetShred

Syntax prototype:

```
xam_status
VIM_XSet_SetShred (const xset_handle inHandle,
                   const xam_boolean inBinding,
                   const xam_boolean inShred);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inShred is a xam_boolean containing a flag indicating if shred is enabled on this XSet or not. If the flag is set to TRUE, shredding is enabled, otherwise it is disabled.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have auto shred set on it, this method will create a property field on the specified XSet with the name of *.xset.deletion.shred* and a type set to "application/vnd.snia.xam.boolean". Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine if the XSet should be automatically shredded when it's deleted.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.6 Storage policy

6.2.7.6.1 VIM_XSet_ApplyStoragePolicy

Syntax prototype:

```
xam_status
VIM_XSet_ApplyStoragePolicy (const xset_handle inHandle,
                             const xam_boolean inBinding,
                             const xam_string inPolicy);
```

Parameters:

- inHandle is a valid xset_handle. This object will contain the new field.
- inBinding is a xam_boolean set to TRUE, if the field should be binding, or FALSE otherwise.
- inPolicy is a xam_string containing the name of the policy to be applied.

Error conditions:

- The first argument is not a valid xset_handle.
- The third argument does not contain the name of a valid policy.
- The XSet was opened in readonly mode.
- The XSet was opened in restricted mode and the second argument is set to TRUE.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The field does not already exist on the XSet, and the maximum number of fields allowed on this XSet has been reached.

Description:

If this XSet does not have a storage policy applied to it, this method will create a property field on the specified XSet with the name of *.xset.storage.policy* and a type set to “application/vnd.snia.xam.string”. Its value and binding attributes will be set according to the user-provided parameters. If the field already exists on the XSet, then its value will be updated with the specified value. This field will be used by the XAM Storage System to determine the storage policy of the XSet.

Note: Changing this field from binding to nonbinding (or vice versa) will result in a new XSet being created and a new XUID being assigned on a successful commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.7 Policy evaluation

6.2.7.7.1 VIM_XSet_GetActualRetentionDuration

Syntax prototype:

```
xam_status
VIM_XSet_GetActualRetentionDuration (const xset_handle inHandle,
                                     const xam_string inRetentionID,
                                     xam_int* const outDuration);
```

Parameters:

- inHandle is a valid xset_handle.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.
- outDuration is a reference to valid storage for a xam_int. On return, this value will be set to the actual minimum retention duration (in milliseconds) that is currently in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention identifier does not exist in the XSet.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a retention duration policy that does not exist.
- The XSet instance was imported and contains a retention duration policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect the retention duration that is currently in effect for the XSet under the scope of the specified retention id and return that duration to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.7.2 VIM_XSet_GetActualRetentionEnabled

Syntax prototype:

```
xam_status
VIM_XSet_GetActualRetentionEnabled (const xset_handle inHandle,
                                    const xam_string inRetentionID,
                                    xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- inRetentionID is a xam_string containing the retention identifier of the retention being created.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument does not contain a validly formatted retention identifier.
- The retention identifier does not exist in the XSet.
- The third argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a retention enabled policy that does not exist.
- The XSet instance was imported and contains a retention enabled policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if retention is enabled for the XSet under the scope of the specified retention id and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.7.3 VIM_XSet_GetActualAutoDelete

Syntax prototype:

```
xam_status
VIM_XSet_GetActualAutoDelete (const xset_handle inHandle,
                              xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains an auto-delete policy that does not exist.
- The XSet instance was imported and contains an auto-delete policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if auto delete is enabled for the XSet and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.7.7.4 VIM_XSet_GetActualShred

Syntax prototype:

```
xam_status
VIM_XSet_GetActualShred (const xset_handle inHandle,
                         xam_boolean* const outEnabled);
```

Parameters:

- inHandle is a valid xset_handle.
- outEnabled is a reference to valid storage for a xam_boolean. On return, this value will be set to match the enabled state in effect for the XSet after evaluating the policies. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The XSet instance was imported and contains a shred policy that does not exist.
- The XSet instance was imported and contains a shred policy that does not match the policy in the XSystem.

Description:

This method will evaluate all factors that affect if shredding is enabled for the XSet and return that enabled state to the caller.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.8 XSet instance administration

When applications invoke XSet instance management methods, the XAM Library must return a non-zero error status when the specified xam_handle_t refers to the XAM_HANDLE or an XSystem, and it shall not invoke VIM methods in these cases. The VIM methods defined in this section shall only be invoked when the application provides an XSet handle.

6.2.8.1 VIM_XSet_Commit

Syntax prototype:

```
xam_status
```

```
VIM_XSet_Commit (const xset_handle inHandle,
                 XUID* outXUID);
```

Parameters:

- inHandle is an xset_handle.
- outXUID is a reference to valid storage for a XUID. On return, this value will contain the XUID that was assigned to the XSet by the XAM Storage System. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument is NULL.
- The XSystem does not have authorization to commit an XSet.
- The XSet is not valid, or has been modified in an invalid way (e.g., a field does not have a valid type).
- The XSet contains a running job (see Section 5.3.10.1, "Jobs"), and the XAM Storage System does not support committing running jobs.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will store an XSet in the XSystem. This does not close the XSet, which can still be modified as allowed by the authorization of the XSystem. A XUID will be assigned by the XAM Storage System and this XUID will be returned.

If this is a modified XSet (e.g., an existing XSet was opened, changed, and then committed), then a new XUID may or may not be assigned, according to the following rules:

- If only variable fields are edited (created, deleted, or changed), then the XAM Storage System may not assign a new XUID.
- If any binding fields are edited (created, deleted, or changed), then the XAM Storage System must assign a new XUID.

In any case, an application should be coded to handle cases where the XUID changes when a modified XSet is committed.

If a management policy has not been applied to the XSet before commit, a default management policy will be applied to the XSet at the time of commit.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.8.2 VIM_XSet_Close

Syntax prototype:

```
xam_status
VIM_XSet_Close (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- There are open XStreams.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is called to release any resources associated with an XSet. After calling this method, the closed XSet should not be used.

Note: This call will fail if there are any open XStreams associated with this XSet.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.8.3 VIM_XSet_Abandon

Syntax prototype:

```
xam_status
VIM_XSet_Abandon (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

An XSet in its normal state will generate an error when an application attempts to close it, if there are open XStreams in it. Making this call will change the state of the XSet and allow it to be closed without regard for any open XStreams.

Note: The XSet will no longer be usable after this call is made, and the only call that will succeed is XSet.close.

CAUTION: This very dangerous call may result in data loss if used inappropriately. It is recommended that applications track all open XStreams and close the XStreams properly as opposed to making this call.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.9 XSet migration

When applications invoke XSet migration methods, the XAM Library must return a non-zero error status when the specified xam_handle_t refers to the XAM_HANDLE or an XSystem, and it shall not invoke VIM methods in these cases. The VIM methods defined in this section shall only be invoked when the application provides an XSet handle.

6.2.9.1 VIM_XSet_OpenExportXStream

Syntax prototype:

```
xam_status
VIM_XSet_OpenExportXStream (const xset_handle inHandle,
                            xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xset_handle.
- outXStream is a reference to valid storage for a xstream_handle. On return, this value will contain the XStream handle of an XStream opened in “r” mode. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSystem does not have authorization to export an XSet.
- The XSet has any open XStreams (including import or export XStreams).
- The XSet is in a corrupt state. (as a result of a failed import).
- The XSet has never been committed.
- The XSet has been modified since it was opened.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an export XStream for the XSet. The XSet must have been committed and must not have been modified since it was opened / committed. The XSet will enter an import/export state, and will, thus, generate errors if used for any operation until the export XStream is closed. The original XSet referred to by the XSet handle will be overwritten.

The XStream will contain a canonical representation of the XSet. This data can be read from the XStream using normal XStream calls and semantics. When the XStream is closed, the XSet will return to a normal state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.9.2 VIM_XSet_OpenImportXStream

Syntax prototype:

```
xam_status
VIM_XSet_OpenImportXStream (const xset_handle inHandle,
                            xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xset_handle.
- outXStream is a reference to valid storage for a xstream_handle. On return, this value will contain the XStream handle of an XStream opened in “w” mode. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is NULL.
- The XSystem does not have authorization to import an XSet.
- The XSet was a not newly created XSet.
- The XSet has been modified since it was created.
- The XSet has any open XStreams (including import or export XStreams).
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will open an import XStream for the XSet. The XSet will enter an import/export state and, therefore, will generate errors if used for any operation until the XStream is closed. The original XSet referred to by the XSet handle will be overwritten.

It is expected that a data stream containing the canonical representation of an XSet will be written into the XStream. When the XStream is closed, the data will be validated. If the data is determined to be valid, then the XSet will return to a normal state (i.e., will no longer generate errors when operated on), but it will now refer to the XSet that was described by the canonical data that was written to the XStream. If the validation of the data fails (i.e., it contains invalid or improperly formatted data), then the XSet will enter a corrupted state. It will no longer be recoverable, and all operations, except XSet.abandon (followed by XSet.close), will fail.

After a successful validation, the XSet fields can be examined as any normal fields. The XSet can be modified. The XSet is not committed, but it is in all ways a normal XSet and may be committed as per normal XSet semantics.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.10 Asynchronous operations

6.2.10.1 Jobs

When applications invoke XSet job methods, the XAM Library must return a non-zero error status when the specified xam_handle_t refers to the XAM_HANDLE or an XSystem, and it shall not invoke VIM methods in these cases. The VIM methods defined in this section shall only be invoked when the application provides an XSet handle.

6.2.10.1.1VIM_XSet_SubmitJob

Syntax prototype:

```
xam_status
VIM_XSet_SubmitJob (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSystem does not have authorization to submit a job.
- The XSet does not contain valid job control fields.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will submit a job request to the XAM Storage System. Fields on the XSet will be evaluated as input to the job, according to the semantics of the XAM job control subsystem (refer to the [XAM-ARCH] for more details). This XSet will be used to communicate health and status information about the job, as well as any results from the job.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.10.1.2VIM_XSet_HaltJob

Syntax prototype:

```
xam_status
VIM_XSet_HaltJob (const xset_handle inHandle);
```

Parameters:

- inHandle is an xset_handle.

Error conditions:

- The first argument is not a valid xset_handle.
- The XSystem does not have authorization to halt a job.

- The XSet is does not contain valid job control fields.
- The XSet was not used to submit a job.

CAUTION: If the XSet has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will stop a currently running job in the XAM Storage System, if the XSet was used to start a job. Fields on the XSet will be evaluated as input to the job, according to the semantics of the XAM job control subsystem (refer to the [XAM-ARCH] for more details).

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

6.2.10.2 XSet async I/O

6.2.10.2.1VIM_XSystem_AsyncOpenXSet

Syntax prototype:

```
xam_status
VIM_XSystem_AsyncOpenXSet (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           const xam_string inMode,
                           const XOPID inXOPID,
                           xasync_callback inCallback,
                           xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be opened.
- inMode is a string indicating the mode to open the XSet in:
  - — readonly: open for reading. Adding, deleting, or modifying fields is not allowed. Commit of the XSet instance will fail.
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail, if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).

- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not readonly, restricted, or unrestricted.
- The XSet is on hold, and the mode is not readonly.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The sixth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous opening of an XSet in the XSystem, ultimately returning a handle to an XSet instance that is associated with the XSystem. The specified callback will be invoked as part of the asynchronous opening. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe. It is the responsibility of the application to ensure that the callback is coded in a thread-safe manner.

Blocking:

This method will not block until complete, and will return control immediately.

6.2.10.2.2VIM_XSystem_AsyncCopyXSet

Syntax prototype:

```
xam_status
VIM_XSystem_AsyncCopyXSet (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           const xam_string inMode,
                           const XOPID inXOPID,
                           xasync_callback inCallback,
                              xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xsystem_handle.
- inXUID is the XUID of the XSet to be copied.
- inMode is a string indicating the mode to open the copied XSet in:
  - — restricted: open for reading and limited writing. Adding, deleting, or modifying fields that are binding is not allowed. Changing fields from binding to nonbinding (or vice versa) is not allowed. Commit of the XSet instance will fail if any binding fields have been modified. Successful commit of the XSet will never generate a new XUID.
  - — unrestricted: open for reading and writing. There are no limits on adding, deleting, or modifying fields or changing fields from binding to nonbinding (or vice versa). Successful commit of the XSet will generate a new XUID, if any binding fields have been added, deleted, or modified, or if any fields have been changed from binding to nonbinding (or vice versa).
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The second argument contains an improperly formatted XUID.
- The third argument is NULL.
- The third argument is not restricted or unrestricted.
- The XSystem does not have authorization to open an XSet.
- The XSet does not exist in the XSystem.
- The sixth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous copying of an XSet in the XSystem, ultimately returning a handle to an XSet instance that is associated with the XSystem. The specified callback will be invoked as part of the asynchronous copying. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe.

6.2.10.2.3VIM_XSet_AsyncOpenXStream

Syntax prototype:

```
xam_status
VIM_XAM_OpenXStream (const xam_handle_t inHandle,
                     const xam_string inName,
                     const xam_string inMode,
                     const XOPID inXOPID,
                     xasync_callback inCallback,
                     xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM object reference. This object will contain the new field.
- inName is a xam_string containing the name of the field to be opened.
- inMode is a string indicating the mode to open the XStream in:
  - — readonly: open for reading. Write methods will fail on the XStream instance.
  - — writeonly: open for writing. Truncates existing data in the XStream. Read and seek methods will fail on the XStream instance.
  - — appendonly: open for writing. Appends to existing data in the XStream. Read and seek methods will fail on the XStream instance.
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xam_handle_t.
- The second argument is not a valid name (invalid UTF-8).
- The second argument contains a name of a field not present.
- The third argument contains anything other than writeonly, appendonly, or readonly.
- The sixth argument is NULL.
- The xam_handle_t contains an XSet that was opened in readonly mode, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, the field is binding, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that was opened in restricted mode, is on hold, and the XStream open mode is writeonly or appendonly.
- The xam_handle_t contains an XSet that is in a corrupt state.
- The xam_handle_t contains an XSet that is in an abandoned state.

- The xam_handle_t contains an XSystem that is in a corrupt state.
- The xam_handle_t contains an XSystem that is in an abandoned state.

Description:

This method will begin the asynchronous opening of XStream in either readonly, writeonly, or appendonly mode, based on the mode argument. The specified callback will be invoked as part of the asynchronous opening. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will not block until complete, and will return control immediately.

6.2.10.2.4VIM_XStream_AsyncRead

Syntax prototype:

```
xam_status
VIMXStream_AsyncRead (const xstream_handle inHandle,
                      char* const ioBuffer,
                      const xam_int inBufferLength,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in readonly mode.
- ioBuffer is a byte array to read the data into.
- inBufferLength is a xam_int set to the number of bytes in the buffer.
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened in writeonly mode.
- The second argument is NULL.
- The buffer length is less than or equal to zero.

- The sixth argument is NULL.

CAUTION: If the inBufferLength is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous transfer of data from the storage system into the target buffer, up to the number of bytes requested. The specified callback will be invoked as part of the asynchronous transfer. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Concurrency requirements:

This method is thread safe.

Blocking:

This method returns immediately.

6.2.10.2.5VIM_XStream_AsyncWrite

Syntax prototype:

```
xam_status
VIM_XStream_AsyncWrite (const xstream_handle inHandle,
                        const char* const inBuffer,
                        const xam_int inByteCount,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                        xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xstream_handle that must have been opened in writeonly mode.
- inBuffer is a byte array containing the data to be written.
- inByteCount is a xam_int set to the number of bytes in the buffer to be written.
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.
- The first argument is an XStream that was opened in readonly mode.

- The second argument is NULL.
- The maximum length (in bytes) of an XStream is exceeded.
- The sixth argument is NULL.

CAUTION: If the inByteCount is set to a size larger than the actual number of bytes of storage available in the inBuffer, undefined results may occur, including data loss and data corruption.

Description:

This method will begin the asynchronous transfer of data from the source buffer to the XAM Storage System, up to the number of bytes requested. The specified callback will be invoked as part of the asynchronous transfer. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Note: This method may fail with an error if the maximum number of bytes supported in an XStream is reached. To determine the actual maximum number of bytes allowed in an XStream, an application should evaluate *.xsystem.limits.maxSizeOfXStream* on the XSystem instance. For more information on this topic, please consult [XAM-ARCH].

Concurrency requirements:

This method is thread safe.

Blocking:

This method returns immediately.

6.2.10.2.6VIM_XStream_AsyncClose

Syntax prototype:

```
xam_status
VIM_XStream_AsyncClose (const xstream_handle inHandleXStream,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                        xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandleXStream is an xstream_handle.
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xstream_handle.

- The sixth argument is NULL.

CAUTION: Closing an already closed XStream can produce undefined results, including data loss and data corruption.

Description:

This method will begin the asynchronous closing of a previously opened XStream. Any resources that were allocated can be released at this point. The specified callback will be invoked as part of the asynchronous close. To monitor the status of this operation, the application can poll the Async instance that is generated by this method. A handle to an Async instance is also passed to any provided callback method when that callback method is invoked.

Note: The application is responsible for tracking the parent of the XStream. The XOPID can be used for this.

Concurrency requirements:

This method is thread safe.

Blocking:

This method returns immediately.

6.2.10.2.7VIM_XSet_AsyncCommit

Syntax prototype:

```
xam_status
VIM_XSet_AsyncCommit (const xset_handle inHandle,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* const outAsyncHandle);
```

Parameters:

- inHandle is an xset_handle.
- inXOPID is an application-assigned id that is used to distinguish this operation from others.
- inCallback is a function to invoke during the asynchronous processing of this method.
- outAsyncHandle is a reference to valid storage for an xasync_handle. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xsystem_handle.
- The XSystem does not have authorization to commit an XSet.
- The XSet that was opened in readonly mode.

- The XSet was opened in restricted mode and one or more binding fields have been created, modified, or deleted, or one or more fields have been changed from binding to nonbinding (or vice versa).
- The XSet is not valid, or has been modified in an invalid way (e.g., a field does not have a valid type).
- The XSet contains a running job (see Section Section 5.3.10.1, “Jobs”) and the XAM Storage System does not support committing running jobs.
- The XSet has an open import or export stream.
- The XSet is in a corrupt state.
- The XSet is in an abandoned state.
- The fourth argument is NULL.

CAUTION: If the XSystem has been closed, undefined results may occur, including data loss and data corruption.

Description:

This method is an asynchronous version of XSet.commit. See Section 5.3.7.1, “XSet_Commit” for additional information.

Concurrency requirements:

This method is thread safe.

Blocking:

This method returns immediately.

6.2.10.3 Asynchronous Operations Management

Asynchronous operations are in one of two states: pending and completed. When the operation is first initiated, it is in the pending state. Because the operation has not completed, it is only possible to query whether the operation has completed, retrieve the XOPID that was specified when the operation was initiated, and to halt the operation.

6.2.10.3.1VIM_XAsync_Halt

Syntax prototype:

```
xam_status
VIM_XAsync_Halt (const xasync_handle inHandle);
```

Parameters:

- inHandle is an xasync_handle.

Error conditions:

- The first argument is not a valid xasync_handle.

Description:

This method stops the execution of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.2VIM_XAsync_IsComplete

Syntax prototype:

```
xam_status
VIM_XAsync_IsComplete (const xasync_handle inHandle,
                       xam_boolean* const outIsComplete);
```

Parameters:

- inHandle is an xasync_handle.
- outIsComplete is a reference to valid storage for a xam_boolean. On return, this value will be set to TRUE if the operation has completed, FALSE otherwise. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.

Description:

This method retrieves the completed state of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.3 VIM_XAsync_GetXOPID

Syntax prototype:

```
xam_status
VIM_XAsync_GetXOPID (const xasync_handle inHandle,
                     XOPID* const outXOPID);
```

Parameters:

- inHandle is an xasync_handle.
- outXOPID is a reference to valid storage for a XOPID. On return, it is set to the value of the XOPID. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the XOPID of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.4 VIM_XAsync_GetStatus

Syntax prototype:

```
xam_status
VIM_XAsync_GetStatus (const xasync_handle inHandle,
                      xam_status* const outStatus);
```

Parameters:

- inHandle is an xasync_handle.
- outStatus is a reference to valid storage for a xam_status. On return, this value will be set to the status if the operation has completed. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation has not transitioned to the completed state.
- The operation was programmatically halted.

Description:

This method retrieves the xam_status of the operation that is associated with the Async instance. It may be used after the operation has transitioned to the completed state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.5VIM_XAsync_GetXSet

Syntax prototype:

```
xam_status
VIM_XAsync_GetXSet (const xasync_handle inHandle,
                    xset_handle* const outXSet);
```

Parameters:

- inHandle is an xasync_handle.
- outXSet is a reference to valid storage for a xam_handle. On return, this value will be set to the xset_handle associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no xset_handle associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xset_handle of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.6 VIM_XAsync_GetXStream

Syntax prototype:

```
xam_status
VIM_XAsync_GetXStream (const xasync_handle inHandle,
                       xstream_handle* const outXStream);
```

Parameters:

- inHandle is an xasync_handle.
- outXStream is a reference to valid storage for a xam_handle. On return, this value will be set to the xstream_handle associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no xstream_handle associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xstream_handle of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.7VIM_XAsync_GetXUID

Syntax prototype:

```
xam_status
VIM_XAsync_GetXUID (const xasync_handle inHandle,
                    xam_xuid* const outXUID);
```

Parameters:

- inHandle is an xasync_handle.
- outXUID is a reference to valid storage for a XUID. On return, this value will be set to the XUID associated with the operation. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- There is no XUID associated with the operation.
- The operation was programmatically halted.

Description:

This method retrieves the xset_handle of the operation that is associated with the Async instance. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.8VIM_XAsync_GetBytesRead

Syntax prototype:

```
xam_status
VIM_XAsync_GetBytesRead (const xasync_handle inHandle,
                         xam_int* const outBytesRead);
```

Parameters:

- inHandle is an xasync_handle.
- outBytesRead is a reference to valid storage for a xam_int. On return, this value will be set to the number of bytes read by the operation or to zero, if no data has been read, or, if the operation does not read bytes. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the number of bytes read by the operation that is associated with the Async instance. Not all operations read bytes, and for those operations, it will always be set to zero. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.9VIM_XAsync_GetBytesWritten

Syntax prototype:

```
xam_status
VIM_XAsync_GetBytesWritten (const xasync_handle inHandle,
                            xam_int* const outBytesWritten);
```

Parameters:

- inHandle is an xasync_handle.
- outBytesWritten is a reference to valid storage for a xam_int. On return, this value will be set to the number of bytes written by the operation or to zero, if no data has been written, or if the operation does not write bytes. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xasync_handle.
- The second argument is NULL.
- The operation was programmatically halted.

Description:

This method retrieves the number of bytes written by the operation that is associated with the Async instance. Not all operations write bytes, and for those operations, it will always be set to zero. It may be used at any time.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

6.2.10.3.10VIM_XAsync_Close

Syntax prototype:

```
xam_status
VIM_XAsync_Close (const xasync_handle inHandle);
```

Parameters:

- inHandle is an xasync_handle.

Error conditions:

- The first argument is not a valid xasync_handle.
- The operation has not transitioned to the completed state.

Description:

This method releases the resources of the operation that is associated with the Async instance and of the Async instance itself. It may be used after the operation has transitioned to the completed state.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will return immediately.

Annex A (normative) Public Header Files

The following section contains header files created according to the public API calls defined above.

A.1 xam_types.h

```
#ifndef XAM_TYPES_H
#define XAM_TYPES_H 1

/* ========================================================================== *\
 *                       Supporting definitions                               *
\* ========================================================================== */

#define XAM_MAX_STRING 512
#define XAM_MAX_XUID 80

#ifndef TRUE
  #define TRUE (unsigned char)1
#endif
#ifndef FALSE
  #define FALSE (unsigned char)0
#endif

#define XAM_INT_MAX   0x7fffffffL

#ifdef _WIN32
  #undef EXPORT
  #define EXPORT __declspec(dllexport)
  #undef DECL
  #define DECL __cdecl
#else
  #define EXPORT
  #define DECL
#endif

/* ========================================================================= *\
 *                           Primitive types                                 *
\* ========================================================================= */


#ifdef _WIN32
  #if defined (__GNUC__)
    #include <stdint.h>
    typedef int64_t xam_int;                    /**< 8-byte signed integer   */
  #else
    typedef __int64 xam_int;                    /**< 8-byte signed integer   */
 #endif
#else // POSIX
   #include <inttypes.h>
   #if defined (__IBMC__) || defined (__IBMCPP__)
     typedef long long xam_int;                 /**< 8-byte signed integer */
   #else
     typedef int64_t xam_int;                   /**< 8-byte signed integer */
   #endif
#endif /* WIN32 or POSIX */
```

```
#if !defined(__cplusplus)
    typedef unsigned char           xam_boolean;
    #define true                    (unsigned char)1
    #define false                   (unsigned char)0
#else
    typedef bool                    xam_boolean;
#endif

typedef double                  xam_double;
typedef char                    xam_xuid[XAM_MAX_XUID];
typedef char                    xam_string[XAM_MAX_STRING];
typedef char                    xam_datetime[XAM_MAX_STRING];

typedef xam_xuid                XUID;

/* ======================================================================== *\
 *                          Method status                                   *
\* ======================================================================== */

#ifdef _WIN32
  #if defined (__GNUC__)
    typedef int32_t xam_status;                  /**< 4-byte signed integer */
  #else
    typedef __int32 xam_status;                  /**< 4-byte signed integer */
 #endif
#else // POSIX
   #if defined (__IBMC__) || defined (__IBMCPP__)
      typedef int xam_status;                    /**< 4-byte signed integer */
   #else
      typedef int32_t xam_status;                /**< 4-byte signed integer */
   #endif
#endif /* WIN32 or POSIX */


/* ======================================================================== *\
 *                  Handles for the various XAM classes                     *
\* ======================================================================== */

typedef xam_int                xam_handle_t;
typedef xam_handle_t           xset_handle;
typedef xam_handle_t           xsystem_handle;
typedef xam_handle_t           xiterator_handle;
typedef xam_handle_t           xstream_handle;
typedef xam_handle_t           xasync_handle;

#define XAM_LIBRARY_HANDLE     (xam_handle_t)1   /**< XAM Library constant */

/* ======================================================================== *\
 *                 asynchronous operations typedefs                         *
\* ======================================================================== */

typedef xam_int                XOPID;

typedef void (*xasync_callback) (const xasync_handle inHandle);

/* ======================================================================== *\
 *                         Misc API constants                               *
\* ======================================================================== */

/* XStream whence values */
```

```
static const xam_int    XSTREAM_SEEK_SET  = 0;
static const xam_int    XSTREAM_SEEK_CUR  = 1;
static const xam_int    XSTREAM_SEEK_END  = 2;

/* Access bits */
static const xam_int    XSET_R_OK    = 0x80000000L;   /* Read */
static const xam_int    XSET_WU_OK   = 0x40000000L;   /* Write-user */
static const xam_int    XSET_WS_OK   = 0x20000000L;   /* Write-system */
static const xam_int    XSET_C_OK    = 0x10000000L;   /* Create */
static const xam_int    XSET_D_OK    = 0x08000000L;   /* Delete */
static const xam_int    XSET_H_OK    = 0x04000000L;   /* Hold */
static const xam_int    XSET_RE_OK   = 0x02000000L;   /* Retention-event */
static const xam_int    XSET_J_OK    = 0x01000000L;   /* Job */
static const xam_int    XSET_JC_OK   = 0x00800000L;   /* Job-commit */

/* Access bit composites */
static const xam_int    XSET_W_OK    = (XSET_WU_OK|XSET_WS_OK);                  /
                                   * Write */
static const xam_int    XSET_ALL_OK  = (XSET_R_OK|XSET_W_OK|XSET_C_OK|XSET_D_OK); /
                                   * Read, Write, Create, and Delete */


#endif /* XAM_TYPES_H */
```

A.2 xam_strings.h

```
#ifndef XAM_STRINGS_H
#define XAM_STRINGS_H 1

/* ==================================================================== *\
 *                      environment variable names                      *
\* ==================================================================== */

static const char* const XAM_VIM_PATH_ENV_VAR    = "XAM_VIM_PATH";

/* ==================================================================== *\
 *                      connection string elements                      *
\* ==================================================================== */

static const char* const XAM_PROTOCOL            = "snia-xam://";
static const char* const XAM_VIM_LIBRARY_TOKEN   = "!";

/* ==================================================================== *\
 *                             stypes                                   *
\* ==================================================================== */

static const char* const XAM_BOOLEAN_MIME_TYPE   = "application/
                                   vnd.snia.xam.boolean";
static const char* const XAM_INT_MIME_TYPE       = "application/vnd.snia.xam.int";
static const char* const XAM_DOUBLE_MIME_TYPE    = "application/
                                   vnd.snia.xam.double";
static const char* const XAM_XUID_MIME_TYPE      = "application/vnd.snia.xam.xuid";
static const char* const XAM_STRING_MIME_TYPE    = "application/
                                   vnd.snia.xam.string";
static const char* const XAM_DATETIME_MIME_TYPE  = "application/
                                   vnd.snia.xam.datetime";

/* ================================================================== *\
 *                  XSet create/open mode strings                     *
```

```
\* ================================================================ */

static const char* const XSET_MODE_READ_ONLY  = "readonly";
static const char* const XSET_MODE_RESTRICTED = "restricted";
static const char* const XSET_MODE_MODIFY     = "unrestricted";

/* ================================================================ *\
 *                   XStream open mode strings                      *
\* ================================================================ */

static const char* const XSTREAM_MODE_READ_ONLY          = "readonly";
static const char* const XSTREAM_MODE_WRITEONLY_TRUNCATE = "writeonly";
static const char* const XSTREAM_MODE_WRITEONLY_APPEND   = "appendonly";

/* ================================================================== *\
 *                         XAM object field names                       *
\* ================================================================== */

static const char* const XAM_IDENTITY           = ".xam.identity";
static const char* const XAM_API_LEVEL          = ".xam.apiLevel";
static const char* const XAM_VIM_LIST           = ".xam.vim.list";
static const char* const XAM_MAX_STRING_FIELD   = ".xam.maxstring";

/* ================================================================== *\
 *                 XAM library logging control field names              *
\* ================================================================== */

static const char* const XAM_LOG_LEVEL          = ".xam.log.level";
static const char* const XAM_LOG_VERBOSITY      = ".xam.log.verbosity";
static const char* const XAM_LOG_PATH           = ".xam.log.path";

/* ================================================================== *\
 *                 XAM library log config field names                   *
\* ================================================================== */

static const char* const XAM_LOG_FORMAT             = ".xam.log.format";
static const char* const XAM_LOG_APPEND             = ".xam.log.append";
static const char* const XAM_LOG_MAX_SIZE           = ".xam.log.max.size";
static const char* const XAM_LOG_MAX_ROLLOVERS      = ".xam.log.max.rollovers";
static const char* const XAM_LOG_MSG_FILTER         = ".xam.log.message.filter";
static const char* const XAM_LOG_COMP_FILTER        = ".xam.log.component.filter";
static const char* const XAM_LOG_CFG_PATH           = ".xam.log.config.path";
static const char* const XAM_LOG_CFG_POLL_INTERVAL  =
                                   ".xam.log.config.path.pollInterval";

/* ================================================================== *\
 *                         XSystem field names                          *
\* ================================================================== */

static const char* const XAM_XSYSTEM_INITIALIZING        = ".xsystem.initializing";

static const char* const XAM_XSYSTEM_IDENTITY             = ".xsystem.identity";
static const char* const XAM_XSYSTEM_TIME                 = ".xsystem.time";
static const char* const XAM_XSYSTEM_MAX_FIELDS           =
                                   ".xsystem.limits.maxFieldsPerXSet";
static const char* const XAM_XSYSTEM_MAX_SIZE_OF_XSTREAM  =
                                   ".xsystem.limits.maxSizeOfXStream";
static const char* const XAM_XSYSTEM_SASL_LIST            =
                                   ".xsystem.auth.SASLmechanism.list";
```

```
static const char* const XAM_XSYSTEM_SASL_DEFAULT         =
                               ".xsystem.auth.SASLmechanism.default";

static const char* const XAM_XSYSTEM_AUTH_GRANULE_LIST                =
                               ".xsystem.auth.granule.list";
static const char* const XAM_XSYSTEM_AUTH_IDENTITY_AUTHENTICATION    =
                               ".xsystem.auth.identity.authentication";
static const char* const XAM_XSYSTEM_AUTH_IDENTITY_AUTHORIZATION     =
                               ".xsystem.auth.identity.authorization";
static const char* const XAM_XSYSTEM_AUTH_EXPIRATION                 =
                               ".xsystem.auth.expiration";

static const char* const XAM_XSYSTEM_ACCESS                          =
                               ".xsystem.access";
static const char* const XAM_XSYSTEM_ACCESS_POLICY_LIST              =
                               ".xsystem.access.policy.list";
static const char* const XAM_XSYSTEM_MANAGEMENT_POLICY_LIST          =
                               ".xsystem.management.policy.list";
static const char* const XAM_XSYSTEM_MANAGEMENT_POLICY_DEFAULT       =
                               ".xsystem.management.policy.default";
static const char* const XAM_XSYSTEM_STORAGE_POLICY_LIST             =
                               ".xsystem.storage.policy.list";

static const char* const XAM_XSYSTEM_JOB_COMMIT_SUPPORTED            =
                               ".xsystem.job.commit.supported";
static const char* const XAM_XSYSTEM_JOB_LIST                        =
                               ".xsystem.job.list";
static const char* const XAM_XSYSTEM_JOB_LIST_QUERY                  =
                               ".xsystem.job.list.xam.job.query";
static const char* const XAM_XSYSTEM_JOB_QUERY_CONTINUANCE_SUPPORTED =
                               ".xsystem.job.xam.job.query.continuance.supporte
                               d";
static const char* const XAM_XSYSTEM_JOB_QUERY_LEVEL1_SUPPORTED      =
                               ".xsystem.job.xam.job.query.level1.supported";
static const char* const XAM_XSYSTEM_JOB_QUERY_LEVEL2_SUPPORTED      =
                               ".xsystem.job.xam.job.query.level2.supported";

static const char* const XAM_XSYSTEM_AUTODELETE                      =
                               ".xsystem.deletion.autodelete";
static const char* const XAM_XSYSTEM_AUTODELETE_POLICY_LIST          =
                               ".xsystem.deletion.autodelete.policy.list";
static const char* const XAM_XSYSTEM_SHRED                           =
                               ".xsystem.deletion.shred";
static const char* const XAM_XSYSTEM_SHRED_POLICY_LIST               =
                               ".xsystem.deletion.shred.policy.list";

static const char* const XAM_XSYSTEM_RETENTION_DURATION_POLICY_LIST  =
                                ".xsystem.retention.duration.policy.list";
static const char* const XAM_XSYSTEM_RETENTION_ENABLED_POLICY_LIST   =
                                ".xsystem.retention.enabled.policy.list ";
static const char* const XAM_XSYSTEM_RETENTION_BASE_ENABLED_POLICY_LIST =
                               ".xsystem.retention.base.enabled.policy.list";
static const char* const XAM_XSYSTEM_RETENTION_BASE_DURATION_POLICY_LIST =
                               ".xsystem.retention.base.duration.policy.list";
static const char* const XAM_XSYSTEM_RETENTION_EVENT_DURATION_POLICY_LIST =
                               ".xsystem.retention.event.duration.policy.list";
static const char* const XAM_XSYSTEM_RETENTION_EVENT_ENABLED_POLICY_LIST =
                               ".xsystem.retention.event.enabled.policy.list";
```

```
/* ================================================================== *\
 *                      XSet basic field names                        *
\* ================================================================== */

static const char* const XAM_XSET_TIME_CREATION      = ".xset.time.creation";
static const char* const XAM_XSET_TIME_XUID          = ".xset.time.xuid";
static const char* const XAM_XSET_TIME_COMMIT        = ".xset.time.commit";
static const char* const XAM_XSET_TIME_ACCESS        = ".xset.time.access";
static const char* const XAM_XSET_TIME_RESIDENCY     = ".xset.time.residency";
static const char* const XAM_XSET_XUID               = ".xset.xuid";
static const char* const XAM_XSET_DIRTY              = ".xset.dirty";

/* ================================================================== *\
 *                   XSet management field names                     *
\* ================================================================== */

static const char* const XAM_XSET_ACCESS_POLICY                =
                                  ".xset.access.policy";
static const char* const XAM_XSET_MANAGEMENT_POLICY            =
                                  ".xset.management.policy";
static const char* const XAM_XSET_STORAGE_POLICY               =
                                  ".xset.storage.policy";

static const char* const XAM_XSET_RETENTION_LIST               =
                                  ".xset.retention.list";
static const char* const XAM_XSET_RETENTION_LIST_BASE          =
                                  ".xset.retention.list.base";
static const char* const XAM_XSET_RETENTION_LIST_EVENT         =
                                  ".xset.retention.list.event";

static const char* const XAM_XSET_RETENTION_BASE_ENABLED         =
                                  ".xset.retention.base.enabled";
static const char* const XAM_XSET_RETENTION_BASE_START_TIME      =
                                  ".xset.retention.base.starttime";
static const char* const XAM_XSET_RETENTION_BASE_DURATION        =
                                  ".xset.retention.base.duration";
static const char* const XAM_XSET_RETENTION_BASE_ENABLED_POLICY  =
                                  ".xset.retention.base.enabled.policy";
static const char* const XAM_XSET_RETENTION_BASE_DURATION_POLICY =
                                  ".xset.retention.base.duration.policy";

static const char* const XAM_XSET_RETENTION_EVENT_ENABLED         =
                                  ".xset.retention.event.enabled";
static const char* const XAM_XSET_RETENTION_EVENT_START_TIME      =
                                  ".xset.retention.event.starttime";
static const char* const XAM_XSET_RETENTION_EVENT_DURATION        =
                                  ".xset.retention.event.duration";
static const char* const XAM_XSET_RETENTION_EVENT_ENABLED_POLICY  =
                                  ".xset.retention.event.enabled.policy";
static const char* const XAM_XSET_RETENTION_EVENT_DURATION_POLICY =
                                  ".xset.retention.event.duration.policy";

static const char* const XAM_XSET_AUTODELETE                   =
                                  ".xset.deletion.autodelete";
static const char* const XAM_XSET_AUTODELETE_POLICY            =
                                  ".xset.deletion.autodelete.policy";
static const char* const XAM_XSET_SHRED                        =
                                  ".xset.deletion.shred";
static const char* const XAM_XSET_SHRED_POLICY                 =
                                  ".xset.deletion.shred.policy";
```

```
static const char* const XAM_XSET_HOLD                        = ".xset.hold";
static const char* const XAM_XSET_HOLD_LIST                   = ".xset.hold.list";

/* ================================================================== *\
 *                  XSet Job Properties/Values field names            *
\* ================================================================== */

static const char* const XAM_JOB_COMMAND                = 
                                  "org.snia.xam.job.command";
static const char* const XAM_JOB_STATUS                 = ".xam.job.status";

static const char* const XAM_JOB_STATUS_NEW             = "NEW";
static const char* const XAM_JOB_STATUS_STARTING        = "STARTING";
static const char* const XAM_JOB_STATUS_RUNNING         = "RUNNING";
static const char* const XAM_JOB_STATUS_SHUTTING_DOWN   = "SHUTTING DOWN";
static const char* const XAM_JOB_STATUS_COMPLETE        = "COMPLETE";
static const char* const XAM_JOB_STATUS_SUSPENDED       = "SUSPENDED";
static const char* const XAM_JOB_STATUS_HALTED          = "HALTED";
static const char* const XAM_JOB_STATUS_KILLED          = "KILLED";

static const char* const XAM_JOB_ERRORHEALTH            = ".xam.job.errorhealth";
static const char* const XAM_JOB_ERRORHEALTH_OK         = "OK";
static const char* const XAM_JOB_ERRORHEALTH_ERROR      = "ERROR";

static const char* const XAM_JOB_ERROR                  = ".xam.job.error";

/* =================================================================== *\
 *                     XSet job command values                         *
\* =================================================================== */

static const char* const XAM_JOB_QUERY                  = "xam.job.query";

/* =================================================================== *\
 *                     XSet Query Job field names                      *
\* =================================================================== */

static const char* const XAM_QUERY_XUID_LIST_MIME_TYPE  = "application/
                                  vnd.snia.query.xuid_list";

static const char* const XAM_JOB_QUERY_COMMAND          = "xam.job.query.command";
static const char* const XAM_JOB_QUERY_RESULTS          = "xam.job.query.results";
static const char* const XAM_JOB_QUERY_RESULTS_COUNT    = 
                                  "xam.job.query.results.count";
static const char* const XAM_JOB_QUERY_LEVEL            = "xam.job.query.level";

static const char* const XAM_JOB_QUERY_LEVEL_1          = 
                                  "org.snia.xam.job.query.level.1";
static const char* const XAM_JOB_QUERY_LEVEL_2          = 
                                  "org.snia.xam.job.query.level.2";

/* =================================================================== *\
 *                       XAM job error tokens                          *
\* =================================================================== */

static const char* const XAM_JOB_ERROR_NOT_A_JOB                =
                                  "org.snia.xam::not_a_job";
static const char* const XAM_QUERY_ERROR_UNSPECIFIED_CMD        =
                                  "org.snia.xam.query::unspecified_command";
```

```
static const char* const XAM_QUERY_ERROR_LEVEL_NOT_SUPPORTED     =
                                  "xam.job.query::level_not_supported";
static const char* const XAM_QUERY_ERROR_INVALID_CMD_SYNTAX      =
                                  "xam.job.query::invalid_command_syntax";
static const char* const XAM_QUERY_ERROR_INSUFFICIENT_PERMISSION =
                                  "xam.job.query::insufficient_permission";
static const char* const XAM_QUERY_ERROR_INSUFFICIENT_RESOURCES  =
                                  "xam.job.query::insufficient_resources";

/* ==================================================================== *\
 *                       XAM error tokens                               *
\* ==================================================================== */

static const char* const XAM_OK_TOKEN                    = "xam/OK";
static const char* const XAM_UNKNOWN_ERROR_TOKEN         = "xam/unknown error";
static const char* const XAM_OUT_OF_MEMORY_TOKEN         = "xam/out of memory";
static const char* const XAM_INVALID_PARAM_TOKEN         = "xam/invalid parameter";
static const char* const XAM_PARAM_NOT_UTF8_TOKEN        = "xam/non-UTF8 parameter";
static const char* const XAM_INVALID_HANDLE_TOKEN        = "xam/invalid handle";
static const char* const XAM_INVALID_MIME_TYPE_TOKEN     = "xam/invalid mime type";
static const char* const XAM_INVALID_XSTREAM_MODE_TOKEN  = "xam/invalid xstream
                                  mode";
static const char* const XAM_INVALID_XRI_TOKEN           = "xam/invalid XRI";
static const char* const XAM_INVALID_XSET_MODE_TOKEN     = "xam/invalid xset mode";
static const char* const XAM_INVALID_FIELD_NAME_TOKEN    = "xam/invalid field name";
static const char* const XAM_VIM_NOT_FOUND_TOKEN         = "xam/vim not found";
static const char* const XAM_VIM_SYMBOL_NOT_FOUND_TOKEN  = "xam/vim symbol not
                                  found";
static const char* const XAM_FIELD_NOT_FOUND_TOKEN       = "xam/field not found";
static const char* const XAM_FIELD_IS_READ_ONLY_TOKEN    = "xam/field is read only";
static const char* const XAM_FIELD_EXISTS_TOKEN          = "xam/field exists";
static const char* const XAM_FIELD_IN_USE_TOKEN          = "xam/field in use";
static const char* const XAM_MAX_FIELDS_EXCEEDED_TOKEN   = "xam/reached maximum
                                  field limit";
static const char* const XAM_FILESYSTEM_ERROR_TOKEN      = "xam/filesystem error";
static const char* const XAM_XSYSTEM_ABANDONED_TOKEN     = "xam/xsystem abandoned";
static const char* const XAM_XSET_ABANDONED_TOKEN        = "xam/xset abandoned";
static const char* const XAM_XSTREAM_ABANDONED_TOKEN     = "xam/xstream abandoned";
static const char* const XAM_XSYSTEM_CORRUPT_TOKEN       = "xam/xsystem corrupted";
static const char* const XAM_XSET_CORRUPT_TOKEN          = "xam/xset corrupted";
static const char* const XAM_XSTREAM_CORRUPT_TOKEN       = "xam/xstream corrupted";
static const char* const XAM_CONNECT_FAILED_TOKEN        = "xam/connection failed";
static const char* const XAM_AUTH_REQUIRED_TOKEN         = "xam/authentication
                                  required";
static const char* const XAM_AUTH_DATA_NEEDED_TOKEN      = "xam/authentication data
                                  needed";
static const char* const XAM_AUTH_FAILED_TOKEN           = "xam/authentication
                                  failed";
static const char* const XAM_BAD_XUID_FORMAT_TOKEN       = "xam/bad xuid format";
static const char* const XAM_XSET_NOT_FOUND_TOKEN        = "xam/xset not found";
static const char* const XAM_PENDING_TOKEN               = "xam/operation pending";
static const char* const XAM_NOT_SUPPORTED_TOKEN         = "xam/operation not
                                  supported";
static const char* const XAM_OPERATION_NOT_ALLOWED_TOKEN = "xam/operation not
                                  allowed";
static const char* const XAM_OBJECT_IN_USE_TOKEN         = "xam/object in use";
static const char* const XAM_NOT_A_JOB_TOKEN             = "xam/not a job";
static const char* const XAM_JOB_INVALID_CMD_TOKEN       = "xam/job command
                                  invalid";
```

```
static const char* const XAM_JOB_INVALID_CMD_SYNTAX_TOKEN = "xam/job invalid command
                                   syntax";
static const char* const XAM_JOB_ABORTED_TOKEN           = "xam/job aborted";
static const char* const XAM_JOB_LEVEL_NOT_SUPPORTED_TOKEN      = "xam/job level not
                                   supported";
static const char* const XAM_JOB_INSUFFICIENT_PERMISSIONS_TOKEN = "xam/job
                                   insufficent permissions";
static const char* const XAM_JOB_INSUFFICIENT_RESOURCES_TOKEN   = "xam/job
                                   insufficent resources";
static const char* const XAM_JOB_RUNNING_TOKEN           = "xam/job already running";
static const char* const XAM_XSET_UNDER_RETENTION_TOKEN  = "xam/xset is under
                                   retention";
static const char* const XAM_XSET_UNDER_HOLD_TOKEN    = "xam/xset is under hold";
static const char* const XAM_XSET_HOLD_ID_IN_USE_TOKEN   = "xam/hold id already in
                                   use";
static const char* const XAM_XSET_INVALID_RETENTION_VALUE_TOKEN  = "xam/value would
                                   shorten effective retention";
static const char* const XAM_INVALID_POLICY_NAME_TOKEN   = "xam/invalid policy
                                   name";

#endif // XAM_STRINGS_H
```

A.3 xam_errors.h

```
#ifndef XAM_ERRORS_H
#define XAM_ERRORS_H 1

/* ====================================================================== *\
 *               standard error codes  returned by the XAM API            *
\* ====================================================================== */

#define XAM_UNKNOWN_ERROR                 1001   /**< An unknown error occured */
#define XAM_OUT_OF_MEMORY                 1002   /**< Out of memory */
#define XAM_INVALID_PARAM                 1003   /**< Encountered an invalid API
                                   parameter */
#define XAM_PARAM_NOT_UTF8               1004   /**< Parameter found not to be UTF-
                                   8 */
#define XAM_INVALID_HANDLE                1005   /**< Invalid handle parameter */
#define XAM_INVALID_MIME_TYPE             1006   /**< Invalid mime type */
#define XAM_INVALID_XSTREAM_MODE          1007   /**< Invalid XStream mode */
#define XAM_INVALID_XRI                   1008   /**< Invalid XRI */
#define XAM_INVALID_XSET_MODE            1009   /**< Invalid operating mode for the
                                   XSet */
#define XAM_INVALID_FIELD_NAME            1010   /**< The specified field name is
                                   invalid */
#define XAM_VIM_NOT_FOUND                 1011   /**< VIM could not be located or
                                   loaded */
#define XAM_VIM_SYMBOL_NOT_FOUND          1012   /**< Required symbol not found in
                                   VIM */
#define XAM_FIELD_NOT_FOUND               1013   /**< Field not found for a given
                                   handle */
#define XAM_FIELD_IS_READ_ONLY            1014   /**< Attempted to write to a read
                                   only field */
#define XAM_FIELD_EXISTS                  1015   /**< Field already exists */
#define XAM_FIELD_IN_USE                  1016   /**< Field in use error */
#define XAM_MAX_FIELDS_EXCEEDED          1017   /**< Too many fields exist in this
                                   object */
#define XAM_FILESYSTEM_ERROR              1018   /**< Filesystem error */
#define XAM_XSYSTEM_ABANDONED            1019   /**< The XSystem instance has been
                                   abandoned */
```

```
#define XAM_XSET_ABANDONED                1020   /**< The XSet instance has been
                                   abandoned */
#define XAM_XSTREAM_ABANDONED            1021   /**< The XStream instance has been
                                   abandoned */
#define XAM_XSYSTEM_CORRUPT              1022   /**< The XSystem instance has been
                                   corrupted */
#define XAM_XSET_CORRUPT                  1023   /**< The XSet instance has been
                                   corrupted */
#define XAM_XSTREAM_CORRUPT              1024   /**< The XStream instance has been
                                   corrupted */
#define XAM_CONNECT_FAILED             1025   /**< Failed to connect to the XSystem
                                   */
#define XAM_AUTH_REQUIRED                 1026   /**< Authentication is required */
#define XAM_AUTH_DATA_NEEDED            1027   /**< Additional authentication data
                                   is required */
#define XAM_AUTH_FAILED                   1028   /**< Authentication failed */
#define XAM_BAD_XUID_FORMAT               1029   /**< Bad XUD format */
#define XAM_XSET_NOT_FOUND                1030   /**< XSet not found */
#define XAM_PENDING                       1031   /**< Asynchronous operation is
                                   pending */
#define XAM_NOT_SUPPORTED               1032   /**< The operation requested is not
                                   supported */
#define XAM_OPERATION_NOT_ALLOWED         1033   /**< Operation not allowed */
#define XAM_OBJECT_IN_USE               1034   /**< This object is currently in use
                                   */
#define XAM_NOT_A_JOB                   1035   /**< The XSet does not contain a job
                                   request */
#define XAM_JOB_INVALID_CMD             1036   /**< The job command is invalid */
#define XAM_JOB_INVALID_CMD_SYNTAX        1037   /**< The job command syntax is
                                   invalid */
#define XAM_JOB_ABORTED                   1038   /**< The job was aborted */
#define XAM_JOB_LEVEL_NOT_SUPPORTED      1039   /**< The job level is insufficent */
#define XAM_JOB_INSUFFICIENT_PERMISSIONS  1040   /**< The job permissions are
                                   insuffient */
#define XAM_JOB_INSUFFICIENT_RESOURCES    1041   /**< The job resources are
                                   insufficent */
#define XAM_JOB_RUNNING                   1042   /**< A job is already running */
#define XAM_XSET_UNDER_RETENTION         1043   /**< The XSet is under retention */
#define XAM_XSET_UNDER_HOLD               1044   /**< The XSet is under hold and
                                   cannot be deleted */
#define XAM_XSET_HOLD_ID_IN_USE          1045   /**< The hold id is already in use
                                   */
#define XAM_XSET_INVALID_RETENTION_VALUE  1046   /**< The specified duration would
                                   shorten the effective retention */
#define XAM_INVALID_POLICY_NAME           1047   /**< Invalid policy name for this
                                   operation */


#endif // XAM_ERRORS_H
```

A.4 xam.h

```
#ifndef XAM_H
#define XAM_H

/* types and definitions */
#include "xam_types.h"
#include "xam_strings.h"
#include "xam_errors.h"
```

```
#ifdef __cplusplus
extern "C" {
#endif

/* ======================================================================
 * methods for evaluating xam_status
 * ======================================================================*/

/** @defgroup Status Status Methods
    @{ */

/**
    Generates an error token from the xam_status. If passed an XSystem
    reference, it will be able to generate error tokens for non-standard
    status. Otherwise, non-standard status will always generate the
    “xam/unknown error” token.

    This method does not require any passed-in XSystem to be authenticated.
    It will also work on an XSystem that is in a corrupted or aborted state .
    It returns TRUE on success, and FALSE on failure.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.


    @param inHandle A valid xam_handle, containing an XSystem or a XAM
                    library object reference.
    @param inStatus A valid xam_status.
    @param outToken A reference to valid storage for a xam_string. The
                    value that is passed in is not used and is overwritten
                    with the result
    @return true if the error token was found and written outToken,
            false otherwise
 */
EXPORT xam_boolean DECL
XAM_GetErrorToken (const xam_handle_t inHandle,
                   const xam_status inStatus,
                   xam_string* outToken);

/** @} */ /* Status functions */

/* ======================================================================
 * method prototypes for the XIterator
 * ======================================================================*/

/** @defgroup XIterator XIterator Methods
    @{ */

/**
    A factory interface, creating an XIterator from an XSet, XSystem, or XAM
    object (e.g. objects that contain fields). This iterator is used to
    discover the field names of fields on the object in scope (e.g. an XSet,
    XSystem, or XAM object). Only those fields whose names begin with the
    distinct bit sequence as specified in the pattern will be included in
    the enumeration.

    Resources associated with the XIterator must be explicitly released.
    Once the resources are released, the XIterator will no longer be valid.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    fields to be enumerated.
    @param inPattern A valid xam_string, containing a valid, null terminated
                     utf-8 byte sequence. The pattern in this xam_string will
                     be used to filter the fields which will be enumerated -
                     those fields that do not being with the specified pattern
                     will not be included in the enumeration. The pattern is
                     very simple - the byte sequence is treated as an explicit
                     prefix, if the beginning of a field name does not match
                     the exact bit sequence of the specified pattern it will be
                     filtered out of the results. All fields are considered to
                     begin with an empty string, thus specifying an empty
                     string in the pattern will result in no fields being
                     filtered.
    @param outIterator A reference to valid storage for an xiterator_handle.
                       The value that is passed in is not used and is
                       overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_OpenFieldIterator (const xam_handle_t inHandle,
                       const xam_string inPattern,
                       xiterator_handle* outIterator);

/**
    Determines if there are more field names available to be read
    from the XIterator using the XIterator_Next method.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle.
    @param outHasNext A reference to valid storage for a xam_boolean.
                    If additional field names may be read from this XIterator,
                    "true" is written here. Otherwise, "false" is written.
                    The value that is passed in is not used and is overwritten
                    with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XIterator_HasNext (const xiterator_handle inHandle,
                   xam_boolean* outHasNext);

/**
    Copies the field name of the field at the current cursor of the iteration
    into the provided storage. The cursor is then advanced to the next field.
    Upon reading past the last field, an empty string will be returned.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle.
    @param outName  A reference to valid storage for a xam_string. The result
                    is the name of the field following the current cursor (e.g.
                    the field name of the field at the current cursor/position
                    in the iteration). The value that is passed in is not used,
                    and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XIterator_Next (const xiterator_handle inHandle,
                xam_string* outName);

/**
    Releases the resources associated with an open XIterator. After this method
    is called, the XIterator may no longer be used.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XIterator_Close (xiterator_handle inHandle);

/** @} */ /* XIterator functions */

/* ========================================================================
 * method prototypes for managing XAM Fields (properties or XStreams)
 * ========================================================================*/

/** @defgroup Fields Field Management Methods
    @{ */

/**
    Sets the xam_boolean value to true if the named field exists in this
    object, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object on which to determine
                    the existence of the named field.
    @param inName A xam_string containing the name of the field to locate.
    @param outContained A reference to valid storage for a xam_boolean. The result
                        is true if the named field is contained in the object;
                        or false otherwise. The value that is passed in is not
                        used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
```

```
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_ContainsField(const xam_handle_t inHandle,
                  const xam_string inName,
                  xam_boolean* outContained);

/**
    Sets the binding attribute of a field to true.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet reference. This
           is the object that contains the named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetFieldAsBinding (const xset_handle inHandle,
                       const xam_string inName);

/**
    Sets the binding attribute of a field to false.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet reference. This
                    is the object that contains the named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetFieldAsNonbinding (const xset_handle inHandle,
                          const xam_string inName);

/**
    Copies the mime-type of the named field into the provided xam_string.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outType A reference to valid storage for a xam_string. The result
                   is the mime-type of the named field in the object. The value
                   that is passed in is not used and is overwritten with the
                   result.
    @return The status code generated by calling this function. Use the
```

```
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetFieldType (const xam_handle_t inHandle,
                  const xam_string inName,
                  xam_string* outType);

/**
    Copies the length of the named field into the provided xam_int.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outLength A reference to valid storage for a xam_int. The result is
                     the number of bytes of the value of the named field in the
                     object. The value that is passed in is not used and is
                     overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetFieldLength (const xam_handle_t inHandle,
                    const xam_string inName,
                    xam_int* outLength);

/**
    Sets the xam_boolean value to true if the binding attribute of the named
    field is true, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outBinding A reference to valid storage for a xam_boolean. The result
                      is true if the binding attribute of the named field is
                      true;
                      or false otherwise. The value that is passed in is not
                         used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetFieldBinding (const xam_handle_t inHandle,
                     const xam_string inName,
                     xam_boolean* outBinding);

/**
    Sets the xam_boolean value to true if the read-only attribute of the named
    field is true, or to false otherwise.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outReadOnly A reference to valid storage for a xam_boolean. The
                       result is true if the read-only attribute of the named
                       field is true; or false otherwise. The value that is
                       passed in is not used and is overwritten with the
                       result.
   @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
*/
EXPORT xam_status DECL
XAM_GetFieldReadOnly (const xam_handle_t inHandle,
                      const xam_string inName,
                      xam_boolean* outReadOnly);

/**
    Removes a field from the XSet.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    named field.
    @param inName A xam_string containing the name of the field to delete.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_DeleteField (const xam_handle_t inHandle,
                 const xam_string inName);

/** @} */ /* Field functions */

/* ========================================================================
 * method prototypes for managing property fields
 * ========================================================================*/

/** @defgroup Property Property Management Methods
    @{ */

/**
    Creates a property field with a type set to
    "application/vnd.snia.xam.boolean" on the object referenced by the passed
    in xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_boolean containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateBoolean (const xam_handle_t inHandle,
                   const xam_string inName,
                   const xam_boolean inBinding,
                   const xam_boolean inValue);

/**
    Creates a property field with a type set to "application/vnd.snia.xam.int"
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_int containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateInt (const xam_handle_t inHandle,
               const xam_string inName,
               const xam_boolean inBinding,
               const xam_int inValue);

/**
    Creates a property field with a type set to "application/vnd.snia.xam.float"
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_double containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateDouble (const xam_handle_t inHandle,
                 const xam_string inName,
                 const xam_boolean inBinding,
                 const xam_double inValue);

/**
    Creates a property field with a type set to "application/vnd.snia.xam.xuid"
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_xuid containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateXUID (const xam_handle_t inHandle,
                const xam_string inName,
                const xam_boolean inBinding,
                const xam_xuid inValue);

/**
    Creates a property field with a type set to
    "application/vnd.snia.xam.string" on the object referenced by the passed in
    xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_string containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
```

```
EXPORT xam_status DECL
XAM_CreateString (const xam_handle_t inHandle,
                  const xam_string inName,
                  const xam_boolean inBinding,
                  const xam_string inValue);

/**
    Creates a property field with a type set to
    "application/vnd.snia.xam.datetime" on the object referenced by the passed
    in xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_datetime containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateDatetime (const xam_handle_t inHandle,
                    const xam_string inName,
                    const xam_boolean inBinding,
                    const xam_datetime inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.boolean" on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being assigned
          to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_boolean containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetBoolean (const xam_handle_t inHandle,
                const xam_string inName,
                const xam_boolean inValue);
```

```
/**
    Changes a property field with a type set to “application/vnd.snia.xam.int”
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

   Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_int containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetInt (const xam_handle_t inHandle,
            const xam_string inName,
            const xam_int inValue);

/**
    Changes a property field with a type set to
    “application/vnd.snia.xam.float” on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_double containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetDouble (const xam_handle_t inHandle,
              const xam_string inName,
              const xam_double inValue);

/**
    Changes a property field with a type set to “application/vnd.snia.xam.xuid”
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_xuid containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetXUID (const xam_handle_t inHandle,
             const xam_string inName,
             const xam_xuid inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.string" on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_string containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetString (const xam_handle_t inHandle,
               const xam_string inName,
               const xam_string inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.datetime" on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.
    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_datetime containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_SetDatetime (const xam_handle_t inHandle,
                 const xam_string inName,
                 const xam_datetime inValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.boolean" on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_boolean. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetBoolean (const xam_handle_t inHandle,
                const xam_string inName,
                xam_boolean* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.int" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_int. The value of the
                    named field is written into this value. The value that is
                    passed in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
```

```
XAM_GetInt (const xam_handle_t inHandle,
            const xam_string inName,
            xam_int* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.float" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_double. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetDouble (const xam_handle_t inHandle,
              const xam_string inName,
              xam_double* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.xuid" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_xuid. The value of
                    the named field is written into this value. The value that
                    is passed
                    in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetXUID (const xam_handle_t inHandle,
             const xam_string inName,
             xam_xuid* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.string" on the object referenced by the passed in
    xam_handle_t.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_string. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetString (const xam_handle_t inHandle,
               const xam_string inName,
               xam_string* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.datetime” on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_datetime. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_GetDatetime (const xam_handle_t inHandle,
                 const xam_string inName,
                 xam_datetime* outValue);

/** @} */ /* Property functions */

/* ======================================================================
 * method prototypes for XStreams and XStream fields
 * ======================================================================*/

/** @defgroup XStream XStream Methods
    @{ */

/**
    Creates an XStream field with a type set to the user defined mime-type on
    the object referenced by the passed in xam_handle_t. Its name, mime-type
    and binding attributes will be set according to the user provided
    parameters. The XStream field is opened in “writeonly” mode.

    @note The value is not set by the method. This method will create an
          XStream with a length of zero – other methods must be used to add
          data to this field.

    @note If the xam_handle_t contains an XSet, this method may fail with an
          error if the maximum number of fields supported on an XSet is
          reached. All XSystems must support at least XXX fields on an XSet.
          However, some XAM storage systems may support more than this. To
          determine the actual maximum number of fields allowed on an XSet an
          application should evaluate the YYY field on the XSystem. For more
          information on this topic please consult the XAM architecture
          document.

    @note Call the XStream_Close() function one done with the outXStream so
          others can use if needed.

    @note Call the XAM_DeleteField() function to release the resources
          associated with the created outXStream.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inType A xam_string that contains the mime-type of the field.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_CreateXStream (const xam_handle_t inHandle,
                   const xam_string inName,
                   const xam_boolean inBinding,
                   const xam_string inType,
                   xstream_handle* outXStream);

/**
    Creates an open XStream in either “readonly” or “writeonly” mode, based on
    the mode argument.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance. Truncates existing data in the XStream.
        o “appendonly”: open for writing. Read and seek methods will fail on the
                        XStream instance. Appends the existing data in the XStream.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_OpenXStream (const xam_handle_t inHandle,
                 const xam_string inName,
                 const xam_string inMode,
                 xstream_handle* outXStream);

/**
    Transfers data from the storage system into the target buffer, up to the
    number of bytes requested.

    @note If the inBufferLength is set to a size larger than the actual number
          of bytes of storage available in the inBuffer, undefined results may
          occur (this includes but is not limited to data loss and data
          corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method does not block until data is completely read, but will
        indicate the amount of data that was read in each call. Subsequent
        calls may be needed to read the remainder of the data.

    @param inHandle An xstream_handle that must have been opened in read mode.
    @param ioBuffer An allocated byte array into which the data will be read.
    @param inBufferLength A xam_int set to the number of bytes available in
                          the ioBuffer.
    @param outBytesRead A reference to valid storage for a xam_int. On return
                        this will contain the actual number of bytes read. This
                        will be less than or equal to the inBufferLength. When
                        there is no more data to be read, a value of -1 will be
                        set.  The value that is passed in is not used and is
                        overwritten with the result.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_Read (const xstream_handle inHandle,
              char* ioBuffer,
              const xam_int inBufferLength,
              xam_int* outBytesRead);

/**
    Transfers data from the source buffer to the XAM storage system, up to the
    number of bytes requested.

    @note This method may fail with an error if the maximum number of bytes
          supported in an XStream is reached. All XSystems must support at
```

```
            least XXX bytes in an XStream. However, some XAM storage systems may
            support more than this. To determine the actual maximum number of
            bytes allowed in an XStream an application should evaluate the YYY
            field on the XSystem. For more information on this topic please
            consult the XAM architecture document.

      @note If the inByteCount is set to a size larger than the actual number of
            bytes of storage available in the inBuffer, undefined results may
            occur (this includes but is not limited to data loss and data
            corruption).

      Concurrency requirements:
          This method is thread-safe.
      Blocking:
          This method does not block until all the data in the buffer is
          completely written, but it will indicate the amount of data that was
          written in each call. Subsequent calls may be needed to write the all
          of the data.

      @param inHandle An xstream_handle that must have been opened in write mode.
      @param inBuffer A byte array containing the data to be written.
      @param inByteCount A xam_int set to the number of bytes in the buffer to be
                         written.
      @param outBytesWritten A reference to valid storage for a xam_int. On
                             return this will contain the actual number of bytes
                             written. This will be less than or equal to the
                             inByteCount. The value that is passed in is not used,
                             and is overwritten with the result.
      @return The status code generated by calling this function. Use the
              XAM_GetErrorToken function to determine the meaning of this value.
   */
  EXPORT xam_status DECL
  XStream_Write (const xstream_handle inHandle,
                 const char* inBuffer,
                 const xam_int inByteCount,
                 xam_int* outBytesWritten);

  /**
      Sets the position indicator for the XStream. The new position, measured in
      bytes, is obtained by adding inOffset bytes to the position specified by
      inWhence. If inWhence is set to 0, 1, or 2, then the offset is relative to
      the start of the XStream, the current position, or end-of-data,
      respectively.

      @note This method can only be used for XStreams opened for read. In
            addition, this method cannot be used to create sparse files. It is an
            error to seek past the end of the data in the XStream, as indicated by
            the field attribute 'length'.

      Concurrency requirements:
          This method is thread-safe.
      Blocking:
          This method will block until complete.

      @param inHandle An xstream_handle that must have been opened in read mode.
      @param inOffset A xam_int containing the number of bytes to change the
                      position by. A positive value moves the cursor forward.
                      A negative value moves the cursor backward.
      @param inWhence A xam_int containing a 0, 1 or 2 (indicating where the
                      offset should be measured from).
```

```
                    The following constants are provided:
                    XSTREAM_SEEK_SET(0) - Seek from the start position
                    XSTREAM_SEEK_CUR(1) - Seek from the current position
                  XSTREAM_SEEK_END(2) - Seek from the end position
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_Seek (const xstream_handle inHandle,
              const xam_int inOffset,
              const xam_int inWhence);

/**
    Obtains the current value of the XStream position indicator.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle.
    @param outPosition A xam_int containing the position in the XStream.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_Tell (const xstream_handle inHandle,
              xam_int* outPosition);

/**
    An XStream in its normal state will generate an error when an application
    attempts to close it if there are open asynchronous operations being
    performed on it. Making this call will change the state of the XStream and
    allow it to be closed without regard for any open asynchronous operations.
    Note that the XStream will no longer be usable after this call is made, and
    the only call that will succeed is an XStream.Close.

    @note This is a VERY DANGEROUS call that may result in data loss if used
          inappropriately. It is recommended that applications track all open
          asynchronous operations, and close the asynchronous operations
          properly as opposed to making this call.

    @note If the XStream has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_Abandon(const xstream_handle inHandle);

/**
    Closes a previously opened XStream. Any resources that were allocated can
    be released at this point.
```

```
        @note Closing an already closed XStream can produce undefined results (this
              includes but is not limited to data loss and data corruption)

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xstream_handle.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XStream_Close (xstream_handle inHandle);

    /** @} */ /* XStream functions */

    /* ========================================================================
     * method prototypes for managing the connection to a XAM storage system
     * ========================================================================*/

    /** @defgroup XSystem XSystem Methods
        @{ */

    /**
        XAM Applications connect to XAM storage systems by calling the 'connect'
        API method in the XAM API, and specifying the XSystem's Uniform Resource
        Identifier (XRI) string as its parameter. It is expected that the XRI will
        be specified by the local storage system administrators, and applications
        should strive to make this easily configured at rum time.

        @note The XSystem is not fully usable until it has been authenticated.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inXRI A xam_string. It contains the XSystem's Uniform Resource
                     Identifier. A BNF of this format is listed below:

                     snia-xam://[vimname!]xsystemname[?param=value[{&param=value}]]

                     The vimname is a string that describes which VIM to use, and
                     if it is not specified the XAM system will choose a VIM to
                     use. A vimname is not allowed to contain a '!' character. The
                     xsystemname is vendor specific – it may be an IP address, or
                     some other id. It may not contain '/', '?', or '!' characters.
                     Finally, param'='value pairs can be specified
        @param outHandle A reference to valid storage for an xsystem_handle.
                         On return this will contain the XSystem handle that was
                         created. The value that is passed in is not used and is
                         overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XAMLibrary_Connect (const xam_string inXRI,
                        xsystem_handle* outHandle);
```

```
/**
    Allows an application to authenticate an XSystem. It provides a generic
    interface to exchange data as part of the authentication process. The
    application should check the XSystem property xyz to determine which
    patterns of authentication (mechanisms) are available for use. After a
    pattern is selected, the appropriate sequence of data exchanges should be
    made (using this call) in order to authenticate. A failed authentication
    will make the XSystem unusable – applications cannot repeat failed
    authentications using the same XSystem.

    @note The outXStream must be closed (using XStream_Close() function) when
          the application has finished its authentication processing.

    @note If the XSystem has been closed, or if the inByteCount is set to a
          size larger than the actual number of bytes of storage available in
          the inBuffer, undefined results may occur (this includes but is not
          limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inBuffer Data that is being passed to the authentication mechanism
                    is passed in this array of bytes.
    @param inByteCount The number of significant bytes in the passed in buffer.
    @param outXStream A reference to valid storage for an xstream_handle. On
                      return this will contain the XStream handle that was
                      created, and which contains the systems response to the
                      authentication information. The value that is passed in
                      is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_Authenticate (const xsystem_handle inHandle,
                      const char* inBuffer,
                      const xam_int inByteCount,
                      xstream_handle* outXStream);

/**
    Called to release any resources associated with an XSystem. After calling
    this method, the closed XSystem should not be used.

    @note This call will fail if there are any open XSets associated with this
          XSystem.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_Close (const xsystem_handle inHandle);

/**
    An XSystem in its normal state will generate an error when an application
    attempts to close it if it has open XSets in it. Making this call will
    change the state of the XSystem and allow it to be closed without regard
    for any open XSets. Note that the XSystem will no longer be usable after
    this call is made, and the only call that will succeed is an XSystem.Close.

    @note This is a VERY DANGEROUS call that may result in data loss if used
          inappropriately. It is recommended that applications track all open
          XSets, and close the XSets properly as opposed to making this call.

   @note If the XSystem has been closed undefined results may occur (this
         includes but is not limited to data loss and data corruption).

   Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_Abandon (const xsystem_handle inHandle);

/** @} */ /* XSystem functions */

/* ===========================================================================
 * method prototypes for XSet management
 * ===========================================================================*/

/** @defgroup XSet XSet Management Methods
    @{ */

/**
    Creates a new empty XSet associated with the XSystem. Note that this XSet
    will not exist on the XSystem unless that XSet is committed.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inMode A string indicating the mode to open the XSet in.
        Possible values are:
        o "restricted" - open for reading and limited writing. Adding, deleting
                         or modifying fields that are binding is not allowed.
                         Changing fields from binding to nonbinding (or vice
                         versa) is not allowed. Commit of the XSet instance
                         will fail if any binding fields havebeen modified.
                         Successful commit of the XSet will never generate a
```

```
                        new XUID.
        o “unrestricted” - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).
    @param outXSet A reference to valid storage for an xset_handle. The value
                   that is passed in is not used and is overwritten with the
                   result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_CreateXSet (const xsystem_handle inHandle,
                    const xam_string inMode,
                    xset_handle* outXSet);

/**
    Opens an XSet in the XSystem.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to open the XSet in:
        o “readonly” - open for reading. Adding, deleting or modifying fields
                      is not allowed. Commit of the XSet instance will fail.
        o “restricted” - open for reading and limited writing. Adding, deleting
                        or modifying fields that are binding is not allowed.
                        Changing fields from binding to nonbinding (or vice
                        versa) is not allowed. Commit of the XSet instance
                        will fail if any binding fields havebeen modified.
                        Successful commit of the XSet will never generate a
                        new XUID.
        o “unrestricted” - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).
    @param outXSet A reference to valid storage for a xset_handle. On return
                   this will contain the XSet handle. The value that is passed
                   in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_OpenXSet (const xsystem_handle inHandle,
                  const xam_xuid inXUID,
                  const xam_string inMode,
                  xset_handle* outXSet);


/**
    Creates a copy of an XSet in the XSystem, returning a handle to an
    XSet instance associated with the XSystem. This XSet will not exist
    on the XSystem unless that XSet instance is committed.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete. For applications that wish
        to use a non-blocking version of this method, refer to
        "XSystem_AsyncCopyXSet".

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to open the XSet in:
        o "restricted" - open for reading and limited writing. Adding, deleting
                         or modifying fields that are binding is not allowed.
                         Changing fields from binding to nonbinding (or vice
                         versa) is not allowed. Commit of the XSet instance
                         will fail if any binding fields havebeen modified.
                         Successful commit of the XSet will never generate a
                         new XUID.
        o "unrestricted" - open for reading and writing. There are no limits
                           on adding, deleting or modifying fields; nor are
                           there limits on changing fields from binding to
                           nonbinding (or vice versa). Successful commit of the
                           XSet will generate a new XUID if any binding fields
                           have been added, deleted, or modified, or if any
                           fields have been changed from binding to nonbinding
                           (or vice versa).
    @param outXSet A reference to valid storage for a xset_handle. On return
                   this will contain the XSet handle. The value that is passed
                   in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_CopyXSet (const xsystem_handle inHandle,
                  const xam_xuid inXUID,
                  const xam_string inMode,
                  xset_handle* outXSet);

/**
    Evaluates all retention criteria that exists on a given XSet, specified
    as a xam_xuid, and shall return TRUE if there exists retention criteria
    which would prohibit XSet deletion. The method returns FALSE if all XSet
    retention criteria have been met. This method does not evaluate the
    on-hold status.

    A non-fatal error will be returned if the specified XUID is improperly
    formatted, does not exist in the XSystem, or if the caller is not
    authorized to access the XSet.
    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be checked.
    @param outIsRetained A reference to valid storage for a xam_boolean.
                         On return this will be set to true if the XSet is
                         under retention in accordance with the XSet retention
                         criteria, false otherwise. The value that is
                         passed in is not used, and is overwritten with the
                         result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_IsXSetRetained (const xsystem_handle inHandle,
                        const xam_xuid inXUID,
                        xam_boolean* const outIsRetained);

/**
    Deletes an XSet from the XSystem.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be deleted.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_DeleteXSet (const xsystem_handle inHandle,
                    const xam_xuid inXUID);

/**
    Places an XSet on hold. A held XSet cannot be changed in any way (e.g. the
    XSet can only be opened in read mode and commits of a held XSet will fail).

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be held.
    @param inHoldID A xam_string that contains the ID to be associated with the
                    hold.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_HoldXSet (const xsystem_handle inHandle,
                  const xam_xuid inXUID,
                  const xam_string inHoldID);

/**
    Releases a specific hold on an XSet (associated with the hold id).

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be held.
    @param inHoldID A xam_string that contains the ID associated with the hold.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_ReleaseXSet (const xsystem_handle inHandle,
                     const xam_xuid inXUID,
                     const xam_string inHoldID);

/**
    Checks the accessibility of an XSet on the XSystem. It is not an error if
    the XSet does not exist on the XSystem: such an XSet shall be noted as
    being inaccessible.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be checked.
    @param inMode The bitwise OR of the access ‘permissions’ to be checked:
                   - XSET_R_OK for read permission
                   - XSET_W_OK for write permission
                   - XSET_D_OK for delete permission
    @param outIsAccessible A reference to valid storage for a xam_boolean.
                           On return this will be set to true if the XSet is
                           accessible according to the access permissions
                           set by mode, false otherwise. The value that is
                           passed in is not used and is overwritten with the
                           result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_AccessXSet (const xsystem_handle inHandle,
                    const xam_xuid inXUID,
                    const xam_int inMode,
                    xam_boolean* outIsAccessible);

/**
    Gets the time at which the XSet was last opened or committed, whichever is
    the most recent.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be checked.
    @param outAccessTime A reference to valid storage for a xam_datetime. On
                         return this will be set to the time at which the XSet
                         was last opened or committed, whichever is the most
                         recent. The value that is passed in is not used and
                         is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_GetXSetAccessTime (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           xam_datetime* outAccessTime);

/** @} */ /* XSet Management Methods */

/* ===============================================================================
 * method prototypes for XSet instance adminstration
 * ==============================================================================*/

/** @defgroup XSetAdmin XSet Instance Administation Methods
    @{ */

/**
    Stores an XSet in the XSystem. Note this does not close the XSet, which can
    still be modified as allowed by the authorization of the XSystem. A XUID
    will be assigned by the XAM storage system and this XUID will be returned.

    Open XStreams will not cause the commit to fail. Only the data that was
    successfully written to such XSteams will be committed.

    If this is a modified XSet (e.g. an existing XSet was opened and changed)
    then a new XUID may or may not be assigned by the commit, in accordance
    with the following rules:
        - If only variable fields are edited (created, deleted, or changed)
          then the XAM storage system may not assign a new XUID.
        - If any binding fields are edited (created, deleted, or changed) then
          the XAM storage system must assign a new XUI.

   In any case, an application should be coded to handle cases where the XUID
   changes when a modified XSet is committed.

   If a management policy has not been applied to the XSet prior to commit, a
```

```
      default management policy will be applied to the XSet at the time of commit.

       @note If the XSystem has been closed undefined results may occur (this
             includes but is not limited to data loss and data corruption).

       Concurrency requirements:
           This method is thread-safe.
       Blocking:
           This method will block until complete.

       @param inHandle An xset_handle.
       @param outXUID A reference to valid storage for a XUID. On return this will
                      contain the XUID that was assigned to the XSet by the XAM
                      storage system. The value that is passed in is not used and
                      is overwritten with the result.
       @return The status code generated by calling this function. Use the
               XAM_GetErrorToken function to determine the meaning of this value.
    */
   EXPORT xam_status DECL
   XSet_Commit (const xset_handle inHandle,
                xam_xuid* outXUID);

   /**
       Releases any resources associated with an XSet. After calling this method,
       the closed XSet should not be used.

       @note This call will fail if there are any open XStreams associated with
             this XSet.

       @note if the XSet has been closed undefined results may occur (this
             includes but is not limited to data loss and data corruption).

       Concurrency requirements:
           This method is thread-safe.
       Blocking:
           This method will block until complete.

       @param inHandle An xset_handle.
       @return The status code generated by calling this function. Use the
               XAM_GetErrorToken function to determine the meaning of this value.
    */
   EXPORT xam_status DECL
   XSet_Close (const xset_handle inHandle);

   /**
       An XSet in its normal state will generate an error when an application
       attempts to close it if there are open XStreams in it. Making this call
       will change the state of the XSet and allow it to be closed without regard
       for any open XStreams. Note that the XSet will no longer be usable after
       this call is made, and the only call that will succeed is an XSet.Close.

       @note this is a VERY DANGEROUS call that may result in data loss if used
             inappropriately. It is recommended that applications track all open
             XStreams, and close the XStreams properly as opposed to making this
             call.

       @note If the XSet has been closed undefined results may occur (this
             includes but is not limited to data loss and data corruption).

       Concurrency requirements:
           This method is thread-safe.
       Blocking:
           This method will block until complete.

       @param inHandle An xset_handle
       @return The status code generated by calling this function. Use the
               XAM_GetErrorToken function to determine the meaning of this value.
    */
   EXPORT xam_status DECL
   XSet_Abandon (const xset_handle inHandle);

   /** @} */ /* XSet Instance Administration Methods */

   /* ========================================================================
    * method prototypes for XSet policy management
    * ========================================================================*/

   /** @defgroup XSetPolicy XSet Policy Management Methods
       @{ */

   /**
       Creates or modifies a property field with the name of
       ".xset.access.policy" and a type set to "application/vnd.snia.xam.string"
       on the object referenced by the passed-in xset_handle. Its value and
       binding attributes will be set according to the user-provided parameters.
       This field will be used by the XAM Storage System to determine the policies
       to use when accessing this XSet..

       @note If an access policy has not been applied to an XSet at the time of
             the initial commit, then the property will be created and set as the
             default access policy of the XSystem (i.e. the first string in the
             XSystem AccessPolicyList).

       @note Changing this policy from binding to nonbinding (or the converse)
             will result in a new XSet being created and a new XUID being
             assigned on commit.

       Concurrency requirements:
           This method is thread-safe.
       Blocking:
           This method will block until complete.

       @param inHandle A valid xset_handle. This is the object that will contain
                       the new field.
       @param inBinding A xam_boolean set to true if the field should be binding;
                        or false otherwise.
       @param inPolicy A xam_string containing the name of the policy to be
                       applied.
       @return The status code generated by calling this function. Use the
               XAM_GetErrorToken function to determine the meaning of this value.
    */
   EXPORT xam_status DECL
   XSet_ApplyAccessPolicy (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_string inPolicy);

   /**
       Removes all access fields from the XSet.

       @note If an access policy has not been applied to an XSet at the time of
```

```
            the initial commit, then the property will be created and set as
            the default access policy of the XSystem (i.e., the first string
            in the XSystem AccessPolicyList).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

    passed-in    @param inHandle A valid xset_handle. This is the object that contains
                        the access field(s).
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ResetAccessFields (const xset_handle inHandle);

    /**
        Creates a property field with the name of "xam.management_policy" and a
        type set to "application/vnd.snia.xam.string" on the object referenced by
        the passed-in xam_handle_t. Its value and binding attributes will be set in
        accordance with the user-provided parameters. This field will be used by
        the XAM storage system to determine the default policies to use when
        managing this XSet.

        @note If a management policy has not been applied to an XSet at the time
              of the initial commit, then the property will be created and set as
              the default management policy of the XSystem (i.e. first string in
              the XSystem ManagementPolicyList).

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyManagementPolicy (const xset_handle inHandle,
                                const xam_boolean inBinding,
                                const xam_string inPolicy);

    /**
        Removes all management fields from the XSet. This includes the
        ".xset.retention.start_time"; because this is a binding field, calling this
        method will always result in a new XUID being assigned to this XSet at the
        next commit.

        @note If a management policy has not been applied to an XSet at the time of
```

```
          the initial commit, then the property will be created and set as the
          default management policy of the XSystem (i.e. first string in the
          XSystem ManagementPolicyList).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_ResetManagementFields (const xset_handle inHandle);

/**
    Creates a scope to for storing and evaluating retention criteria. It
    creates a field with a type of "application/vnd.snia.xam.string" and
    sets the value to the retention id. The field name is formed by appending
    the retention id to the following prefix: ".xset.retention.list.". Thus
    the final format of the name is .xset.retention.list.<retention id>. It
    will have its binding attribute set according to the binding flag
    set by the application.

    @note Changing this field from binding to nonbinding (or vice versa)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_CreateRetention (const xset_handle inHandle,
                      const xam_boolean inBinding,
                      const xam_string inRetentionID);

/**
    Enables or disables retention that is scoped by the specified retention id.
    This flag is stored in a field of type "application/vnd.snia.xam.boolean".
    The name of the field is formed by inserting the retention id between a
    prefix (.xset.retention.) and a suffix (.enabled); thus, the final format
    of the name is .xset.retention.<retention id>.enabled. If the field does
    not exist it will be created; otherwise the value will be updated if and
    only if the value is changed from false to true - if the value is set to
    true it cannot be changed. It will have its binding attribute set in
    accordance with the binding flag that is set by the application.
```

```
        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inEnabled A xam_boolean containing a flag indicating if event
                         retention is enabled on this XSet or not. If the flag is
                         set to true, event retention is enabled, otherwise it is
                         disabled.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetRetentionEnabledFlag (const xset_handle inHandle,
                                  const xam_string inRetentionID,
                                  const xam_boolean inBinding,
                                  const xam_boolean inEnabled);

    /**
        This method will enabled or disable retention that is scopedretention that is
                                      scoped by the specified
        retention id. The policy name of the policy holding the enabled flag
        is stored in a field of type "application/vnd.snia.xam.string". The name
        of the field is formed by inserting the retention id between a prefix
        (.xset.retention.) and a suffix (.enabled.policy); thus, the final format
        of the name is .xset.retention.<retention id>.enabled.policy. If the field
        does not exist it will be created; otherwise the value will be updated if
        and only if the value is changed from false to true - if the value is set
        to true it cannot be changed. It will have its binding attribute set in
        accordance with the binding flag that is set by the application.

        @note If the .xset.retention.<retention id>.enabled field is also present
              on the XSet, it will be used by the XAM Storage System in preference
              to this field.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyRetentionEnabledPolicy (const xset_handle inHandle,
                                      const xam_string inRetentionID,
                                      const xam_boolean inBinding,
                                      const xam_string inPolicy);

    /**
        Sets the duration of retention that is scoped by the specified retention id.
        This flag is stored in a field of type "application/vnd.snia.xam.int".
        The name of the field is formed by inserting the retention id between
        a prefix (.xset.retention.) and a suffix (.duration); thus, the final
        format of the name is .xset.retention.<retention id>.duration. If the
        field does not exist it will be created; otherwise the value will be
        updated if and only if the duration is increased. It will have its
        binding attribute set according to the binding flag that is set by the
        application.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inDuration A xam_int containing the amount of time (measured in
                          milliseconds from the time of commit) to retain the XSet.
                          Zero indicates no retention, while a negative one (-1)
                          indicates infinite retention.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetRetentionDuration (const xset_handle inHandle,
                               const xam_string inRetentionID,
                               const xam_boolean inBinding,
                               const xam_int inDuration);

    /**
       Sets the duration of retention that is scoped by the specified retention id. This
        policy name is stored in a field of type "application/vnd.snia.xam.string".
        The name of the field is formed by inserting the retention id between a
        prefix (.xset.retention.) and a suffix (.duration.policy); thus, the final
        format of the name is .xset.retention.<retention id>.duration.policy. If
        the field does not exist it will be created; otherwise the value will be
        updated if and only if the duration is increased. It will have its binding
        attribute set according to the binding flag that is set by the application.
        @note If the .xset.retention.<retention id>.duration field is also present
              on the XSet, it will be used by the XAM Storage System in preference
              to this field.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyRetentionDurationPolicy (const xset_handle inHandle,
                                       const xam_string inRetentionID,
                                       const xam_boolean inBinding,
                                       const xam_string inPolicy);
    /**
        Sets the start time of retention that is scoped by the specified retention id.
        The current time of the XSystem is stored in a field of type
        "application/vnd.snia.xam.datetime". The name of the field is formed by
        inserting the retention id between a prefix (.xset.retention.) and a suffix
        (.starttime); thus, the final format of the name is
        .xset.retention.<retention id>.starttime. If the field does not exist, it
        will be created; otherwise, an error will be generated, as it is not allowed
        to change the starttime once set. It will have its binding attribute
        set according to the binding flag that is set by the application.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetRetentionStarttime (const xset_handle inHandle,
                                const xam_string inRetentionID,
                                const xam_boolean inBinding);

    /**
        If this XSet does not already contain the field .xset.retention.list.base,
        this method will create the field with a type of
        “application/vnd.snia.xam.string” and set the value to “base”. It will also
        create the “application/vnd.snia.xam.boolean” field
        .xset.retention.base.enabled and set the value to true. The duration will
        be stored in a field named .xset.retention.base.duration. This field is of
        type “application/vnd.snia.xam.int”. If the field already exists, its value
        will be changed to match the passed-in duration if and only if the duration
        of the retention is not reduced; the method will generate an error if the
        duration is reduced. If the field does not already exist, it will be created
        with the specified duration as the value. These fields will have their
        binding attribute set according to the binding flag that is set by the
        application. These fields will be used by the XAM Storage System to
        determine the base retention duration to use when managing this XSet.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        @note When an XSet instance containing the field .xset.retention.list.base
              is first committed, the field .xset.retention.base.starttime will be
              created and have its value set to .xset.time.xuid.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inDuration A xam_int containing the amount of time (measured in
                          milliseconds from the time of commit) to retain the XSet.
                          Zero indicates no retention, while a negative one (-1)
                          indicates infinite retention.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetBaseRetention (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_int inDuration);

    /**
        If this XSet does not already contain the field .xset.retention.list.base,
        this method will create the field with a type of
        “application/vnd.snia.xam.string” and set the value to “base”. It will also
        create the “application/vnd.snia.xam.boolean” field
        .xset.retention.base.enabled and set the value to true. The duration policy
        will be stored in a field named .xset.retention.base.duration.policy. This
        field is of type “application/vnd.snia.xam.string”. If the field already
        exists, its value will be changed to match the passed-in policy if and only
        if the policy would not reduce the duration of the retention; the method
        will generate an error if the policy reduces the duration. If the field
        does not already exist, it will be created with the specified policy name
```

```
        as the value. These fields will have their binding attribute set in
        accordance with the binding flag that is set by the application. These fields
        will be used by the XAM Storage System to determine the base retention duration
        to use when managing this XSet.

        @note If the .xset.retention.base.duration field is also present on the
              XSet, it will be used by the XAM Storage System in preference to
              this policy field.

        @note When an XSet instance containing the field .xset.retention.list.base
              is first committed, the field .xset.retention.base.starttime will be
              created and have its value set to .xset.time.xuid.

        @note Changing this field from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyBaseRetentionPolicy (const xset_handle inHandle,
                                   const xam_boolean inBinding,
                                   const xam_string inPolicy);

    /**
        Creates a property field on the specified XSet with the name of
        "xam.autodelete_policy" and a type set to "application/vnd.snia.xam.string"
        Its value and binding attributes will be set according to the user
        provided parameters. This field will be used by the XAM storage system to
        determine if the XSet should be automatically deleted upon expiration of
        retention. Applying the policy will also remove the "xam.autodelete" from
        the XSet.

        @note If the explicit duration field is also present on the XSet
              ("xam.autodelete") it will be used by the XAM storage system in
              preference to this field.

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyAutoDeletePolicy (const xset_handle inHandle,
                                const xam_boolean inBinding,
                                const xam_string inPolicy);

    /**
        Creates a property field on the specified XSet with the name of
        "xam.autodelete" and a type set to "application/vnd.snia.xam.boolean". Its
        value and binding attributes will be set according to the user
        provided parameters. This field will be used by the XAM storage system to
        determine if the XSet should be automatically deleted upon expiration of
        retention. Applying the policy will also remove the "xset.autodelete_policy"
        field from the XSet.

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being assigned
              on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inAutoDelete A xam_boolean containing a flag indicating if
                            autodelete is enabled on this XSet or not. If the flag
                            is set to true, autodelete is enabled, otherwise it is
                            disabled.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetAutoDelete (const xset_handle inHandle,
                        const xam_boolean inBinding,
                        const xam_boolean inAutoDelete);

    /**
        If this XSet does not have an auto shred policy applied to it, this method
        will create a property field on the specified XSet with the name of
        ".xset.deletion.shred.policy" and a type set to "application/
                                       vnd.snia.xam.string".
        Its value and binding attributes will be set according to the user-provided
        parameters. If the field already exists on the XSet, then its value will be
        updated with the specified value. This field will be used by the XAM Storage
        System to determine if the XSet should be shredded after XSet deletion.
        If the ".xset.deletion.shred" field is also present on the XSet it will be
        used by the XAM Storage System in preference to this field.

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
```

```
            assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyShredPolicy (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_string inPolicy);

    /**
        If this XSet does not have auto shred set on it, this method will create a
        property field on the specified XSet with the name of
        ".xset.deletion.shred" and a type set to "application/vnd.snia.xam.boolean".
        Its value and binding attributes will be set according to the user-provided
        parameters. If the field already exists on the XSet, then its value will be
        updated with the specified value. This field will be used by the XAM Storage
        System to determine if the XSet should be shredded after deletion.

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being assigned
              on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inShred A xam_boolean containing a flag indicating if shredding is
                       enabled on this XSet or not. If the flag is set to TRUE,
                       shredding is enabled, otherwise it is disabled.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_SetShred (const xset_handle inHandle,
                   const xam_boolean inBinding,
                   const xam_boolean inShred);

    /**
        Creates a property field on the specified XSet with the name of
        "xam.storage_policy" and a type set to "application/vnd.snia.xam.string".
        Its value and binding attributes will be set according to the user
        provided parameters. This field will be used by the XAM storage system to
        determine the storage policy of the XSet.
        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_ApplyStoragePolicy (const xset_handle inHandle,
                             const xam_boolean inBinding,
                             const xam_string inPolicy);

    /**
        Evaluates all factors that affect the retention duration that is
        currently in effect for the given retention id, and returns that
        duration to the caller.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inRetentionID A xam_string containing the retention identifier of the
                             retention being checked.
        @param outDuration A reference to valid storage for a xam_int. On return
                           this will be set to the actual minimum retention duration
                           that is currently being in effect for the XSet after
                           evaluating the policies. The value that is passed in is
                           not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_GetActualRetentionDuration (const xset_handle inHandle,
                                     const xam_string inRetentionID,
                                     xam_int* outDuration);

    /**
        Evaluates all factors that affect if this retention is enabled for the
        XSet, and return that enabled state to the caller.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.
        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param outEnabled A reference to valid storage for a xam_boolean. On return
                          this will be set to match the enabled state in effect for
                          the XSet after evaluating the policies. The value that is
                          passed in is not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_GetActualRetentionEnabled (const xset_handle inHandle,
                                    const xam_string inRetentionID,
                                    xam_boolean* outEnabled);

    /**
        Evaluates all factors that affect if auto delete is enabled for the XSet,
        and return that enabled state to the caller.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param outEnabled A reference to valid storage for a xam_boolean. On return
                          this will be set to match the enabled state in effect for
                          the XSet after evaluating the policies. The value that is
                          passed in is not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XSet_GetActualAutoDelete (const xset_handle inHandle,
                              xam_boolean* outEnabled);

    /**
        Evaluates all factors that affect if shredding is enabled for the XSet and
        return that enabled state to the caller.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param outEnabled A reference to valid storage for a xam_boolean. On return
                          this will be set to match the enabled state in effect for
                          the XSet after evaluating the policies. The value that is
                          passed in is not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    xam_status
    XSet_GetActualShred (const xset_handle inHandle,
                         xam_boolean* outEnabled);

    /** @} */ /* XSet policy management */
```

```
/* ============================================================================
 * method prototypes for XSet migration
 * ============================================================================*/

/** @defgroup Migration Migration Methods
    @{ */

/**
    Opens an export XStream for the XSet. The XSet must have been committed,
    and must not have been modified since it was opened / committed. The XSet
    will enter an import/export state, and will thus generate errors if used
    for any operation until the export XStream is closed. The original XSet
    referred to by the XSet handle will be overwritten.

    The XStream will contain a canonical representation of the XSet. This data
    can be read from the XStream using normal XStream calls and semantics. When
    the XStream is closed the XSet will return to a normal state.


    @note If the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xset_handle.
    @param outXStream A reference to valid storage for a xstream_handle. On
                      return this will contain the XStream handle of an XStream
                      opened in “readonly” mode. The value that is passed in is
                      not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_OpenExportXStream (const xset_handle inHandle,
                        xstream_handle* outXStream);

/**
    Opens an import XStream for the XSet. The XSet will enter an import/export
    state, and will thus generate errors if used for any operation until the
    XStream is closed. Any data in the original XSet instance will be
    overwritten.

    It is expected that a data stream containing the canonical representation
    of an XSet will be written into the XStream. When the XStream is closed,
    the data will be validated. If the data is determined to be valid, then the
    XSet will return to a normal state (i.e. will no longer generate errors
    when operated on) but it will now refer to the XSet that was described by
    the canonical data that was written to the XStream. If the validation of the
    data fails (i.e. it contains invalid or improperly formatted data) then the
    XSet will enter a corrupted state. It will no longer be recoverable and all
    operations except abandon (followed by close) will fail.

    After a successful validation, the XSet fields can be examined as any
    normal fields. The XSet can be modified. The XSet is not committed, but it
    is in all ways a normal XSet, and may be committed as per normal XSet
    semantics. If the XSet is committed prior to any modification to binding
    fields (adding, modifying or deleting binding fields; or changing the
    binding attribute of any fields) then the XUID will be the XUID described
    by the import XStream. Modification to any binding fields as decribed above
    will result in a new XUID being assigned upon commit.

    @note If the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle an xset_handle.
    @param outXStream A reference to valid storage for a xstream_handle. On
                      return this will contain the XStream handle of an XStream
                      opened in “w” mode. The value that is passed in is not
                      used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_OpenImportXStream (const xset_handle inHandle,
                        xstream_handle* outXStream);

/** @} */ /* Migration functions */

/* ==========================================================================
 * method prototypes for job control
 * ==========================================================================*/

/** @defgroup Jobs Job Methods
    @{ */

/**
    Submits a job request to the XAM storage system. Fields on the XSet will be
    evaluated as input to the job according to the semantics of the XAM
    job control subsystem (refer to the XAM architecture document for more
    details). This XSet will be used to communicate health and status
    information about the job, as well as any results from the job.

    @note If the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xset_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_SubmitJob (const xset_handle inHandle);

/**
    Stops a currently running job in XAM storage system, if the XSet was used
    to start a job. Fields on the XSet will be evaluated as input to the job in
    accordance with the semantics of the XAM job control subsystem (refer to
    the XAM architecture document for more details).
    @note If the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xset_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_HaltJob (const xset_handle inHandle);

/** @} */ /* Job functions */

/* ========================================================================
 * method prototypes for async i/o
 * ========================================================================*/

/** @defgroup XAsync Async I/O Methods
    @{ */

/**
    Asynchronously opens an XSet in the XSystem.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to open the XSet in:
       o "readonly" - open for reading. Adding, deleting or modifying fields
                      is not allowed. Commit of the XSet instance will fail.
       o "restricted" - open for reading and limited writing. Adding, deleting
                        or modifying fields that are binding is not allowed.
                        Changing fields from binding to nonbinding (or vice
                        versa) is not allowed. Commit of the XSet instance
                        will fail if any binding fields havebeen modified.
                        Successful commit of the XSet will never generate a
                        new XUID.
       o "unrestricted" - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSystem_AsyncOpenXSet (const xsystem_handle inHandle,
                       const xam_xuid inXUID,
                       const xam_string inMode,
                       const XOPID inXOPID,
                       xasync_callback inCallback,
                       xasync_handle* outAsyncHandle);

/**
    Begins the asynchronous copying of an XSet in the XSystem, ultimately
    returning a handle to an XSet instance associated with the XSystem.
    The specified callback will be invoked as part of the asynchronous
    copying. To monitor the status of this operation, the application can
    poll the Async instance that is generated by this method. A handle to
    an XAsync instance is also passed to any provided callback method when
    that callback method is invoked.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to copy the XSet in:
       o "restricted" - open for reading and limited writing. Adding, deleting
                        or modifying fields that are binding is not allowed.
                        Changing fields from binding to nonbinding (or vice
                        versa) is not allowed. Commit of the XSet instance
                        will fail if any binding fields havebeen modified.
                        Successful commit of the XSet will never generate a
                        new XUID.
       o "unrestricted" - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
xam_status
XSystem_AsyncCopyXSet (const xsystem_handle inHandle,
                       const xam_xuid inXUID,
                      const xam_string inMode,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* outAsyncHandle);

/**
    Asynchronously creates an open XStream instance in either “readonly”
    or “writeonly” mode, based on the mode argument.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that will contain the
                    new field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAM_AsyncOpenXStream (const xam_handle_t inHandle,
                      const xam_string inName,
                      const xam_string inMode,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* outAsyncHandle);

/**
    Asynchronously transfers data from the storage system into the target
    buffer, up to the number of bytes requested.

    @note If the inBufferLength is set to a size larger than the actual
          number of bytes of storage available in the inBuffer, undefined
          results may occur (this includes but is not limited to data loss and
          data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle that must have been opened in read mode.
    @param ioBuffer A byte array to read the data into.
    @param inBufferLength A xam_int set to the number of bytes in the buffer.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
```

```
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_AsyncRead (const xstream_handle inHandle,
                   char* ioBuffer,
                   const xam_int inBufferLength,
                   const XOPID inXOPID,
                   xasync_callback inCallback,
                   xasync_handle* outAsyncHandle);

/**
    Asynchronously transfers data from the source buffer to the XAM storage
    system, up to the number of bytes requested.

    @note This method may fail with an error if the maximum number of bytes
          supported in an XStream is reached. All XSystems must support at
          least XXX bytes in an XStream. However, some XAM storage systems may
          support more than this. To determine the actual maximum number of
          bytes allowed in an XStream an application should evaluate the YYY
          field on the XSystem. For more information on this topic please
          consult the XAM architecture document.

    @note If the inByteCount is set to a size larger than the actual number of
          bytes of storage available in the inBuffer, undefined results may
          occur (this includes but is not limited to data loss and data
          corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle that must have been opened in write mode.
    @param inBuffer A byte array containing the data to be written.
    @param inByteCount A xam_int set to the number of bytes in the buffer to be
                       written.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XStream_AsyncWrite (const xstream_handle inHandle,
                    const char* inBuffer,
                    const xam_int inByteCount,
                    const XOPID inXOPID,
                    xasync_callback inCallback,
                    xasync_handle* outAsyncHandle);

/**
    Asynchronously stores an XSet in the XSystem. Note this does not close
    the XSet, which can still be modified as allowed by the authorization
    of the XSystem. A XUID will be assigned by the XAM storage system and this
    XUID will be returned.

    Open XStreams will not cause the commit to fail. Only the data that was
    successfully written to such XSteams will be committed.

    If this is a modified XSet (e.g. an existing XSet was opened and changed)
    then a new XUID may or may not be assigned by the commit, in accordance
    with the following rules:
        - If only variable fields are edited (created, deleted, or changed)
          then the XAM storage system may not assign a new XUID.
        - If any binding fields are edited (created, deleted, or changed) then
          the XAM storage system must assign a new XUI.

   In any case, an application should be coded to handle cases where the XUID
   changes when a modified XSet is committed.

   If a management policy has not been applied to the XSet prior to commit, a
   default management policy will be applied to the XSet at the time of commit.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xset_handle.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_AsyncCommit (const xset_handle inHandle,
                  const XOPID inXOPID,
                  xasync_callback inCallback,
                  xasync_handle* outAsyncHandle);

/**
    Asynchronously closes a previously opened XStream.
    Any resources that were allocated can be released at this point.

    @note Closing an already closed XStream can produce undefined results (this
          includes but is not limited to data loss and data corruption)

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
```

```
                        asynchronous operation completes. The parameter passed to
                        the call back function can be probed for information.
        @param outAsyncHandle A handle to the asynchronous operation.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XStream_AsyncClose (const xstream_handle inHandle,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                        xasync_handle* outAsyncHandle);

    /** @} */ /* Async functions */

    /* ==========================================================================
     * method prototypes for managing asynchronous operations
     * ==========================================================================*/

    /** @defgroup XAsyncManagement Async Operation Management Methods
        @{ */

    /**
        Stops the operation associated with the passed inHandle

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xasync_handle as retrieved by calling anyone of the
                        XXX_AsynchXXX functions
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XAsync_Halt (const xasync_handle inHandle);

    /**
        Allows the caller to discover if the asynchronous operation relating to the
        passed inHandle is complete or not.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xasync_handle as retrieved by calling anyone of the
                        XXX_AsynchXXX functions
        @param outIsComplete A reference to valid storage for a xam_boolean. The
                             result is true if the async operation related to the
                             passed inHandle is complete,
                             or false otherwise.
                             The value that is passed in is not used and is
                             overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    XAsync_IsComplete (const xasync_handle inHandle,
                       xam_boolean* outIsComplete);
```

```
/**
    Gets the status of the completed asynchronous operation that relates
    to the passed inHandle.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions
    @param outStatus A reference to valid storage for a xam_status.
           On input this param is not used, on output this param is populated
           with the status of the completed asynchronous operation that relates
           to the passed inHandle.

           If the underlying asynchronous operation is not complete
           this function will fail and return a status for this call which
           relates to the failure.

    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetStatus (const xasync_handle inHandle,
                  xam_status* outStatus);

/**
    Gets the XOPID that was set by the application for the asynchronous
    operation that relates to the passed inHandle

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXOPID A reference to valid storage for a XOPID.
                    On input this param is not used.
                    On output (if function is successful) this param is
                              populated with the XOPID of the asynchronous
                              operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetXOPID (const xasync_handle inHandle,
                 XOPID* outXOPID);

/**
    Gets the XSet of the completed asynchronous operation that relates to the
    passed inHandle. The return status of this function is set appropriately on
    success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
```

```
          asynchronous read or this function will not be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXSet A reference to valid storage for an xset_handle.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the XSet of the asynchronous
                             operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetXSet (const xasync_handle inHandle,
                xset_handle* outXSet);

/**
    Gets the XStream from the completed asynchronous operation that relates to the
    passed inHandle. The return status of this function is set appropriately on
    success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXStream A reference to valid storage for an xstream_handle.
                      On input this param is not used,
                      On output (if function is successful) this param is
                                populated with the XStream from the asynchronous
                                operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetXStream (const xasync_handle inHandle,
                   xstream_handle* outXStream);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.xuid" on the object referenced by the passed
    inHandle.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXUID A reference to valid storage for a xam_xuid.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the xam_xuid of the asynchronous
                             operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetXUID (const xasync_handle inHandle,
                xam_xuid* outXUID);

/**
    Gets the number of bytes read from the completed asynchronous operation
    that relates to the passed inHandle. The return status of this function
    is set appropriately on success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    @note The asynchronous operation that relates to the passed inHandle must
          be completed for this function call to be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outBytesRead A reference to valid storage for a xam_int.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the number of bytes read
                             during the asynchronous operation that relates to
                             the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetBytesRead (const xasync_handle inHandle,
                     xam_int* outBytesRead);

/**
    Gets the number of bytes written for the completed asynchronous operation
    that relates to the passed inHandle. The return status of this function
    is set appropriately on success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous write or this function will not be successful.

    @note The asynchronous operation that relates to the passed inHandle must
          be completed for this function call to be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
```

```
                    XXX_AsynchXXX functions.
    @param outBytesWritten A reference to valid storage for a xam_int.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the number of bytes written
                             during the asynchronous operation that relates to
                             the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_GetBytesWritten (const xasync_handle inHandle,
                        xam_int* outBytesWritten);

/**
    Releases resources associated with the completed asynchronous operation
    that relates to the passed inHandle.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XAsync_Close (const xasync_handle inHandle);

/** @} */ /* Async management functions */

#ifdef __cplusplus
} //extern "C"
#endif

#endif // XAM_H
```

Annex B (normative) Private (VIM) Header Files

The following section contains header files created according to the private API calls defined above.

B.1 vim.h

```
#ifndef __VIM_H_
#define __VIM_H_

#include "xam_types.h"
#include "xam_strings.h"
#include "xam_errors.h"

#ifdef __cplusplus
extern "C" {
#endif

/**
    Generates an error token from the xam_status. If passed an XSystem
    reference, it will be able to generate error tokens for non-standard
    status. Otherwise, non-standard status will always generate the
    “xam/unknown error” token.

    This method does not require any passed in XSystem to be authenticated.
    It will also work on an XSystem that is in a corrupted or aborted state .
    It returns TRUE on success, and FALSE on failure.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.


    @param inHandle A valid xam_handle, containing an XSystem or a XAM
                    library object reference.
    @param inStatus A valid xam_status.
    @param outToken A reference to valid storage for a xam_string. The
                    value that is passed in is not used and is overwritten
                    with the result
    @return true if the error token was found and written outToken,
            false otherwise
 */
EXPORT xam_boolean DECL
VIM_XSystem_GetErrorToken (const xsystem_handle inHandle,
                           const xam_status inStatus,
                           xam_string* outToken);

/**********************************************************************
 *  method prototypes for the XIterator
 *********************************************************************/

/**
    A factory interface, creating an XIterator from an XSystem. This iterator
    is used to discover the field names of fields on the XSystem. Only those
    fields whose names begin with the distinct bit sequence as specified in the
    pattern will be included in the enumeration.
    Resources associated with the XIterator must be explicitly released. Once
    the resources are released, the XIterator will no longer be valid.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    fields to be enumerated.
    @param inPattern A valid xam_string, containing a valid, null terminated
                     utf-8 byte sequence. The pattern in this xam_string will
                     be used to filter the fields which will be enumerated -
                     those fields that do not being with the specified pattern
                     will not be included in the enumeration. The pattern is
                     very simple - the byte sequence is treated as an explicit
                     prefix, if the beginning of a field name does not match
                     the exact bit sequence of the specified pattern it will be
                     filtered out of the results. All fields are considered to
                     begin with an empty string, thus specifying an empty
                     string in the pattern will result in no fields being
                     filtered.
    @param outIterator A reference to valid storage for an xiterator_handle.
                       The value that is passed in is not used and is
                       overwritten with the result.
    @return The status code generated by calling this function. Use the
        XAM_GetErrorToken function to determine the meaning of this value.
*/
EXPORT xam_status DECL
VIM_XSystem_OpenFieldIterator (const xsystem_handle inHandle,
                               const xam_string inPattern,
                               xiterator_handle* outIterator);

/**
    A factory interface, creating an XIterator from an XSet. This iterator is
    used to discover the field names of fields on the XSystem. Only those
    fields whose names begin with the distinct bit sequence as specified in the
    pattern will be included in the enumeration.

    Resources associated with the XIterator must be explicitly released. Once
    the resources are released, the XIterator will no longer be valid.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete

    @param inHandle A valid xam_handle_t, containing an XSet, XSystem, or XAM
                    Object reference. This is the object that contains the
                    fields to be enumerated.
    @param inPattern A valid xam_string, containing a valid, null terminated
                     utf-8 byte sequence. The pattern in this xam_string will
                     be used to filter the fields which will be enumerated -
                     those fields that do not being with the specified pattern
                     will not be included in the enumeration. The pattern is
                     very simple - the byte sequence is treated as an explicit
                     prefix, if the beginning of a field name does not match
                     the exact bit sequence of the specified pattern it will be
                    filtered out of the results. All fields are considered to
                    begin with an empty string, thus specifying an empty
                    string in the pattern will result in no fields being
                    filtered.
    @param outIterator A reference to valid storage for an xiterator_handle.
                      The value that is passed in is not used and is
                      overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_OpenFieldIterator (const xset_handle inHandle,
                            const xam_string inPattern,
                            xiterator_handle* outIterator);

/**
    Determines if there are more field names available to be read
    from the XIterator using the VIM_XIterator_Next method.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle.
    @param outHasNext A reference to valid storage for a xam_boolean.
                     If additional field names may be read from this XIterator,
                     "true" is written here. Otherwise, "false" is written.
                     The value that is passed in is not used and is overwritten
                     with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XIterator_HasNext (const xiterator_handle inHandle,
                      xam_boolean* outHasNext);

/**
    Copies the field name of the field at the current cursor of the iteration
    into the provided storage. The cursor is then advanced to the next field.
    Upon reading past the last field, an empty string will be returned.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle.
    @param outName  A reference to valid storage for a xam_string. The result
                   is the name of the field following the current cursor (e.g.
                   the field name of the field at the current cursor/position
                   in the iteration). The value that is passed in is not used,
                   and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XIterator_Next (const xiterator_handle inHandle,
                   xam_string* outName);
```

```
/**
    Releases the resources associated with an open XIterator. After this method
    is called, the XIterator may no longer be used.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xiterator_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XIterator_Close (xiterator_handle inHandle);

/**********************************************************************
 *  method prototypes for managing XAM Fields (properties or XStreams)
 **********************************************************************/

/**
    Sets the xam_boolean value to true if the named field exists in this
    object, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing a valid XSystem reference.
                    This is the object on which to determine the existence
                    of the named field.
    @param inName A xam_string containing the name of the field to locate.
    @param outContained A reference to valid storage for a xam_boolean. The result
                        is true if the named field exists in the object;
                        or false otherwise. The value that is passed in is not
                        used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_ContainsField(const xsystem_handle inHandle,
                          const xam_string inName,
                          xam_boolean* outContained);

/**
    Sets the xam_boolean value to true if the named field exists in this
    object, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing a valid XSet reference.
                    This is the object on which to determine the existence
                    of the named field.
    @param inName A xam_string containing the name of the field to locate.
    @param outExists A reference to valid storage for a xam_boolean. The result
                     is true if the named field exists in the object;
                     or false otherwise. The value that is passed in is not
                     used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ContainsField(const xset_handle inHandle,
                       const xam_string inName,
                       xam_boolean* outExists);

/**
    Sets the binding attribute of a field to true.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet reference. This
           is the object that contains the named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetFieldAsBinding (const xset_handle inHandle,
                            const xam_string inName);

/**
    Sets the binding attribute of a field to false.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet reference. This
           is the object that contains the named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetFieldAsNonbinding (const xset_handle inHandle,
                               const xam_string inName);


/**
    Copies the mime-type of the named field into the provided xam_string.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that contains the named
                    field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outType A reference to valid storage for a xam_string. The result
```

```
                    is the mime-type of the named field in the object. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetFieldType (const xsystem_handle inHandle,
                          const xam_string inName,
                          xam_string* outType);

/**
    Copies the mime-type of the named field into the provided xam_string.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object reference.
                     This is the object that contains the named field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outType A reference to valid storage for a xam_string. The result
                    is the mime-type of the named field in the object. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetFieldType (const xset_handle inHandle,
                       const xam_string inName,
                       xam_string* outType);

/**
    Copies the length of the named field into the provided xam_int.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                     reference. This is the object that contains the named
                     field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outLength A reference to valid storage for a xam_int. The result is
                      the number of bytes of the value of the named field in the
                      object. The value that is passed in is not used and is
                      overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetFieldLength (const xsystem_handle inHandle,
                            const xam_string inName,
                            xam_int* outLength);
/**
    Copies the length of the named field into the provided xam_int.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that contains the named
                    field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outLength A reference to valid storage for a xam_int. The result is
                     the number of bytes of the value of the named field in the
                     object. The value that is passed in is not used and is
                     overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetFieldLength (const xset_handle inHandle,
                         const xam_string inName,
                         xam_int* outLength);

/**
    Sets the xam_boolean value to true if the binding attribute of the named
    field is true, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that contains the named
                    field.
    @param inName A xam_string containing the name of the field to manipulate.
    @param outBinding A reference to valid storage for a xam_boolean. The result
                      is true if the binding attribute of the named field is
                      true;
                      or false otherwise. The value that is passed in is not
                         used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetFieldBinding (const xset_handle inHandle,
                          const xam_string inName,
                          xam_boolean* outBinding);

/**
    Sets the xam_boolean value to true if the binding attribute of the named
    field is true, or to false otherwise.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that contains the named
                    field.
    @param inName A xam_string containing the name of the field to manipulate.
```

```
        @param outReadOnly A reference to valid storage for a xam_boolean. The
                          result is true if the read only attribute of the named
                          field is true; or false otherwise. The value that is
                          passed in is not used and is overwritten with the
                          result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_GetFieldReadOnly (const xsystem_handle inHandle,
                                  const xam_string inName,
                                  xam_boolean* outReadOnly);

    /**
        Sets the xam_boolean value to true if the read-only attribute of the named
        field is true, or to false otherwise.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete

        @param inHandle A valid xset_handle, containing an XSet Object reference.
                        This is the object that contains the named field.
        @param inName A xam_string containing the name of the field to manipulate.
        @param outReadOnly A reference to valid storage for a xam_boolean. The
                           result is true if the read-only attribute of the named
                           field is true; or false otherwise. The value that is
                           passed in is not used and is overwritten with the
                           result.
      @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
    */
    EXPORT xam_status DECL
    VIM_XSet_GetFieldReadOnly (const xset_handle inHandle,
                               const xam_string inName,
                               xam_boolean* outReadOnly);

    /**
        Removes a field from the XSet.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xsystem_handle, containing an XSystem Object
                        reference. This is the object that contains the named
                        field.
        @param inName A xam_string containing the name of the field to delete.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_DeleteField (const xsystem_handle inHandle,
                             const xam_string inName);

    /**
        Removes a field from the XSet.
        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle, containing an XSet Object
                        reference. This is the object that contains the named
                        field.
        @param inName A xam_string containing the name of the field to delete.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_DeleteField (const xset_handle inHandle,
                          const xam_string inName);

    /**********************************************************************
     *  method prototypes for managing property fields
     **********************************************************************/

    /**
        Creates a property field with a type set to
        "application/vnd.snia.xam.boolean" on the object referenced by the passed
        in xam_handle_t. Its name, value and binding attributes will be set in
        accordance with the user-provided parameters.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xsystem_handle, containing an XSystem Object
                        reference. This is the object that will contain the new
                        field.
        @param inName A xam_string containing the name of the field to be created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inValue A xam_boolean containing the value to be stored.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_CreateBoolean (const xsystem_handle inHandle,
                               const xam_string inName,
                               const xam_boolean inBinding,
                               const xam_boolean inValue);

    /**
        Creates a property field with a type set to
        "application/vnd.snia.xam.boolean" on the object referenced by the passed
        in xam_handle_t. Its name, value and binding attributes will be set in
        accordance with the user-provided parameters.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle, containing an XSet Object
                        reference. This is the object that will contain the new
                    field.
        @param inName A xam_string containing the name of the field to be created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
        @param inValue A xam_boolean containing the value to be stored.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_CreateBoolean (const xset_handle inHandle,
                            const xam_string inName,
                            const xam_boolean inBinding,
                            const xam_boolean inValue);

    /**
        Creates a property field with a type set to "application/vnd.snia.xam.int"
        on the object referenced by the passed in xam_handle_t. Its name, value and
        binding attributes will be set according to the user provided
        parameters.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
        @param inName A xam_string containing the name of the field to be created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
        @param inValue A xam_int containing the value to be stored.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_CreateInt (const xsystem_handle inHandle,
                           const xam_string inName,
                           const xam_boolean inBinding,
                           const xam_int inValue);

    /**
        Creates a property field with a type set to "application/vnd.snia.xam.int"
        on the object referenced by the passed in xam_handle_t. Its name, value and
        binding attributes will be set according to the user provided
        parameters.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
        @param inName A xam_string containing the name of the field to be created.
        @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
        @param inValue A xam_int containing the value to be stored.
        @return The status code generated by calling this function. Use the
```

```
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateInt (const xset_handle inHandle,
                    const xam_string inName,
                    const xam_boolean inBinding,
                    const xam_int inValue);
/**
    Creates a property field with a type set to "application/vnd.snia.xam.float"
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_double containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_CreateDouble (const xsystem_handle inHandle,
                          const xam_string inName,
                          const xam_boolean inBinding,
                          const xam_double inValue);

/**
    Creates a property field with a type set to "application/vnd.snia.xam.float"
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_double containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateDouble (const xset_handle inHandle,
                       const xam_string inName,
                       const xam_boolean inBinding,
                       const xam_double inValue);
```

```
/**
    Creates a property field with a type set to “application/vnd.snia.xam.xuid”
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_xuid containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_CreateXUID (const xsystem_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inBinding,
                        const xam_xuid inValue);

/**
    Creates a property field with a type set to “application/vnd.snia.xam.xuid”
    on the object referenced by the passed in xam_handle_t. Its name, value and
    binding attributes will be set according to the user provided
    parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_xuid containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateXUID (const xset_handle inHandle,
                     const xam_string inName,
                     const xam_boolean inBinding,
                     const xam_xuid inValue);

/**
    Creates a property field with a type set to
    “application/vnd.snia.xam.string” on the object referenced by the passed in
    xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_string containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_CreateString (const xsystem_handle inHandle,
                          const xam_string inName,
                          const xam_boolean inBinding,
                          const xam_string inValue);

/**
    Creates a property field with a type set to
    “application/vnd.snia.xam.string” on the object referenced by the passed in
    xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_string containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateString (const xset_handle inHandle,
                       const xam_string inName,
                       const xam_boolean inBinding,
                       const xam_string inValue);

/**
    Creates a property field with a type set to
    “application/vnd.snia.xam.datetime” on the object referenced by the passed
    in xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.
    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_datetime containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_CreateDatetime (const xsystem_handle inHandle,
                            const xam_string inName,
                            const xam_boolean inBinding,
                            const xam_datetime inValue);

/**
    Creates a property field with a type set to
    "application/vnd.snia.xam.datetime" on the object referenced by the passed
    in xam_handle_t. Its name, value and binding attributes will be set in
    accordance with the user-provided parameters.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inValue A xam_datetime containing the value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateDatetime (const xset_handle inHandle,
                         const xam_string inName,
                         const xam_boolean inBinding,
                         const xam_datetime inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.boolean" on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being assigned
          to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_boolean containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetBoolean (const xsystem_handle inHandle,
                        const xam_string inName,
                        const xam_boolean inValue);

/**
    Changes a property field with a type set to
    “application/vnd.snia.xam.boolean” on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being assigned
          to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_boolean containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetBoolean (const xset_handle inHandle,
                     const xam_string inName,
                     const xam_boolean inValue);

/**
    Changes a property field with a type set to “application/vnd.snia.xam.int”
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

   Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_int containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetInt (const xsystem_handle inHandle,
                        const xam_string inName,
                        const xam_int inValue);

/**
    Changes a property field with a type set to “application/vnd.snia.xam.int”
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

   Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_int containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetInt (const xset_handle inHandle,
                           const xam_string inName,
                           const xam_int inValue);

/**
    Changes a property field with a type set to
    “application/vnd.snia.xam.float” on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_double containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetDouble (const xsystem_handle inHandle,
                                const xam_string inName,
                                const xam_double inValue);

/**
    Changes a property field with a type set to
    “application/vnd.snia.xam.float” on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_double containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetDouble (const xset_handle inHandle,
                              const xam_string inName,
                              const xam_double inValue);


/**
    Changes a property field with a type set to "application/vnd.snia.xam.xuid"
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_xuid containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetXUID (const xsystem_handle inHandle,
                                const xam_string inName,
                                const xam_xuid inValue);

/**
    Changes a property field with a type set to "application/vnd.snia.xam.xuid"
    on the object referenced by the passed in xam_handle_t. Its value will be
    set according to the user-provided parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_xuid containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetXUID (const xset_handle inHandle,
                        const xam_string inName,
                        const xam_xuid inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.string" on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_string containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetString (const xsystem_handle inHandle,
                            const xam_string inName,
                            const xam_string inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.string" on the object referenced by the passed in
    xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_string containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetString (const xset_handle inHandle,
                              const xam_string inName,
                              const xam_string inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.datetime" on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_datetime containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_SetDatetime (const xsystem_handle inHandle,
                                  const xam_string inName,
                                  const xam_datetime inValue);

/**
    Changes a property field with a type set to
    "application/vnd.snia.xam.datetime" on the object referenced by the passed
    in xam_handle_t. Its value will be set according to the user provided
    parameter.

    @note If the field is binding, this will result in a new XUID being
          assigned to the XSet upon commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inValue A xam_datetime containing the new value to be stored.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
```

```
VIM_XSet_SetDatetime (const xset_handle inHandle,
                                const xam_string inName,
                                const xam_datetime inValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.boolean” on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_boolean. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetBoolean (const xsystem_handle inHandle,
                                  const xam_string inName,
                                  xam_boolean* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.boolean” on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_boolean. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetBoolean (const xset_handle inHandle,
                               const xam_string inName,
                               xam_boolean* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.int” on the object referenced by the passed in
    xam_handle_t.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_int. The value of the
                    named field is written into this value. The value that is
                    passed in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetInt (const xsystem_handle inHandle,
                              const xam_string inName,
                              xam_int* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.int" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_int. The value of the
                    named field is written into this value. The value that is
                    passed in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetInt (const xset_handle inHandle,
                           const xam_string inName,
                           xam_int* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.float" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_double. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetDouble (const xsystem_handle inHandle,
                                const xam_string inName,
                                xam_double* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.float" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_double. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetDouble (const xset_handle inHandle,
                             const xam_string inName,
                             xam_double* outValue);

/**
    Gets the value from a property field with a type set to
    "application/vnd.snia.xam.xuid" on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_xuid. The value of
                    the named field is written into this value. The value that
                    is passed
                    in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
```

```
VIM_XSystem_GetXUID (const xsystem_handle inHandle,
                                const xam_string inName,
                                xam_xuid* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.xuid” on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_xuid. The value of
                    the named field is written into this value. The value that
                    is passed
                     in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetXUID (const xset_handle inHandle,
                             const xam_string inName,
                             xam_xuid* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.string” on the object referenced by the passed in
    xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_string. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetString (const xsystem_handle inHandle,
                                  const xam_string inName,
                                  xam_string* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.string” on the object referenced by the passed in
    xam_handle_t.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_string. The value of
                    the named field is written into this value. The value that
                    is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetString (const xset_handle inHandle,
                              const xam_string inName,
                              xam_string* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.datetime” on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_datetime. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_GetDatetime (const xsystem_handle inHandle,
                                  const xam_string inName,
                                  xam_datetime* outValue);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.datetime” on the object referenced by the passed
    in xam_handle_t.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param outValue A reference to valid storage for a xam_datetime. The value
                    of the named field is written into this value. The value
                    that is passed in is not used and is overwritten with the
                    result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetDatetime (const xset_handle inHandle,
                                const xam_string inName,
                                xam_datetime* outValue);

/*********************************************************************
 *  method prototypes for XStreams and XStream fields
 ********************************************************************/

/**
    Creates an XStream field with a type set to the user defined mime-type on
    the object referenced by the passed in xam_handle_t. Its name, mime-type
    and binding attributes will be set according to the user provided
    parameters. The XStream field is opened in “writeonly” mode.

    @note The value is not set by the method. This method will create an
          XStream with a length of zero – other methods must be used to add
          data to this field.

    @note If the xam_handle_t contains an XSet, this method may fail with an
          error if the maximum number of fields supported on an XSet is
          reached. All XSystems must support at least XXX fields on an XSet.
          However, some XAM storage systems may support more than this. To
          determine the actual maximum number of fields allowed on an XSet an
          application should evaluate the YYY field on the XSystem. For more
          information on this topic please consult the XAM architecture
          document.

    @note Call the XStream_Close() function one done with the outXStream so
          others can use if needed.

    @note Call the XAM_DeleteField() function to release the resources
          associated with the created outXStream.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inType A xam_string that contains the mime-type of the field.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_CreateXStream (const xsystem_handle inHandle,
                                     const xam_string inName,
                                     const xam_boolean inBinding,
                                     const xam_string inType,
                           xstream_handle* outXStream);

/**
    Creates an XStream field with a type set to the user defined mime-type on
    the object referenced by the passed in xam_handle_t. Its name, mime-type
    and binding attributes will be set according to the user provided
    parameters. The XStream field is opened in “writeonly” mode.

    @note The value is not set by the method. This method will create an
          XStream with a length of zero – other methods must be used to add
          data to this field.

    @note If the xam_handle_t contains an XSet, this method may fail with an
          error if the maximum number of fields supported on an XSet is
          reached. All XSystems must support at least XXX fields on an XSet.
          However, some XAM storage systems may support more than this. To
          determine the actual maximum number of fields allowed on an XSet an
          application should evaluate the YYY field on the XSystem. For more
          information on this topic please consult the XAM architecture
          document.

    @note Call the XStream_Close() function one done with the outXStream so
          others can use if needed.

    @note Call the XAM_DeleteField() function to release the resources
          associated with the created outXStream.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inType A xam_string that contains the mime-type of the field.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateXStream (const xset_handle inHandle,
                                  const xam_string inName,
                                  const xam_boolean inBinding,
                                  const xam_string inType,
                        xstream_handle* outXStream);

/**
    Creates an open XStream in either “readonly” or “writeonly” mode, based on
    the mode argument.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xsystem_handle, containing an XSystem Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance.
        o “appendonly”: open for writing. Read and seek methods will fail on the
                        XStream instance. Appends the existing data in the XStream.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_OpenXStream (const xsystem_handle inHandle,
                                   const xam_string inName,
                                   const xam_string inMode,
                                   xstream_handle* outXStream);

/**
    Creates an open XStream in either “readonly” or “writeonly” mode, based on
    the mode argument.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle, containing an XSet Object
                    reference. This is the object that will contain the new
                    field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance.
        o “appendonly”: open for writing. Read and seek methods will fail on the
                        XStream instance. Appends the existing data in the XStream.
    @param outXStream A reference to valid storage for an xstream_handle. The
                      value that is passed in is not used and is overwritten
                      with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_OpenXStream (const xset_handle inHandle,
                                const xam_string inName,
                                const xam_string inMode,
                                xstream_handle* outXStream);

/**
    Transfers data from the storage system into the target buffer, up to the
    number of bytes requested.

    @note If the inBufferLength is set to a size larger than the actual number
          of bytes of storage available in the inBuffer, undefined results may
          occur (this includes but is not limited to data loss and data
          corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method does not block until data is completely read, but will
        indicate the amount of data that was read in each call. Subsequent
        calls may be needed to read the remainder of the data.

    @param inHandle An xstream_handle that must have been opened in read mode.
    @param ioBuffer A byte array to read the data into.
    @param inBufferLength A xam_int set to the number of bytes in the buffer.
    @param outBytesRead A reference to valid storage for a xam_int. On return
                        this will contain the actual number of bytes read. This
                        will be less than or equal to the inBufferLength. When
                        there is no more data to be read, a value of -1 will be
                        set.  The value that is passed in is not used and is
                        overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Read (const xstream_handle inHandle,
                          char* ioBuffer,
                          const xam_int inBufferLength,
                          xam_int* outBytesRead);

/**
    Transfers data from the source buffer to the XAM storage system, up to the
    number of bytes requested.

    @note This method may fail with an error if the maximum number of bytes
          supported in an XStream is reached. All XSystems must support at
          least XXX bytes in an XStream. However, some XAM storage systems may
          support more than this. To determine the actual maximum number of
          bytes allowed in an XStream an application should evaluate the YYY
          field on the XSystem. For more information on this topic please
          consult the XAM architecture document.

    @note If the inByteCount is set to a size larger than the actual number of
          bytes of storage available in the inBuffer, undefined results may
          occur (this includes but is not limited to data loss and data
          corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method does not block until all the data in the buffer is
        completely written, but it will indicate the amount of data that was
        written in each call. Subsequent calls may be needed to write the all
        of the data.
    @param inHandle An xstream_handle that must have been opened in write mode.
    @param inBuffer A byte array containing the data to be written.
    @param inByteCount A xam_int set to the number of bytes in the buffer to be
                       written.
    @param outBytesWritten A reference to valid storage for a xam_int. On
                           return this will contain the actual number of bytes
                           written. This will be less than or equal to the
                           inByteCount. The value that is passed in is not used,
                           and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Write (const xstream_handle inHandle,
                              const char* inBuffer,
                              const xam_int inByteCount,
                              xam_int* outBytesWritten);

/**
    Sets the position indicator for the XStream. The new position, measured in
    bytes, is obtained by adding inOffset bytes to the position specified by
    inWhence. If inWhence is set to 0, 1, or 2, then the offset is relative to
    the start of the XStream, the current position, or end-of-data,
    respectively.

    @note This method can only be used for XStreams opened for read. In
          addition, this method cannot be used to create sparse files. It is an
          error to seek past the end of the data in the XStream, as indicated by
          the field attribute ‘length’.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle that must have been opened in read mode.
    @param inOffset A xam_int containing the number of bytes to change the
                    position by. A positive value moves the cursor forward.
                    A negative value moves the cursor backward.
    @param inWhence A xam_int containing a 0, 1 or 2 (indicating where the
                    offset should be measured from).
                    The following constants are provided:
                    XSTREAM_SEEK_SET(0) - Seek from the start position
                    XSTREAM_SEEK_CUR(1) - Seek from the current position
                    XSTREAM_SEEK_END(2) - Seek from the end position
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Seek (const xstream_handle inHandle,
                             const xam_int inOffset,
                             const xam_int inWhence);

/**
    Obtains the current value of the XStream position indicator.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle.
    @param outPosition A xam_int containing the position in the XStream.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Tell (const xstream_handle inHandle,
                              xam_int* outPosition);

/**
    An XStream in its normal state will generate an error when an application
    attempts to close it if there are open asynchronous operations being
    performed on it. Making this call will change the state of the XStream and
    allow it to be closed without regard for any open asynchronous operations.
    Note that the XStream will no longer be usable after this call is made, and
    the only call that will succeed is an XStream.Close.

    @note This is a VERY DANGEROUS call that may result in data loss if used
          inappropriately. It is recommended that applications track all open
          asynchronous operations, and close the asynchronous operations
          properly as opposed to making this call.

    @note If the XStream has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Abandon (const xstream_handle inHandle);

/**
    Closes a previously opened XStream. Any resources that were allocated can
    be released at this point.

    @note Closing an already closed XStream can produce undefined results (this
          includes but is not limited to data loss and data corruption)

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xstream_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_Close (xstream_handle inHandle);

/**********************************************************************
 *  Managing the connection to the XAM Storage System
 ***********************************************************************/

/**
    XAM Applications connect to XAM storage systems by calling the 'connect'
    API method in the XAM API, and specifying the XSystem's Uniform Resource
    Identifier (XRI) string as its parameter. It is expected that the XRI will
    be specified by the local storage system administrators, and applications
    should strive to make this easily configured at rum time.

    The XAM Library utilizes this method to create a VIM specific XSystem
    instance handle on which fields may be created. The connection to the
    storage system does not occur until the XAM Library calls the
    VIM_XSystem_Connect method on this handle.

    @note The XSystem is not fully usable until it has been connected
          and authenticated.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param outHandle  A reference to valid storage for an xsystem_handle. On
                      return this will contain the XSystem handle that was
                      created, on which fields may be created/updated.
                      The value that is passed in is not used and is
                      overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_CreateXSystem (xsystem_handle* outHandle);

/**
    XAM Applications connect to XAM storage systems by calling the 'connect'
    API method in the XAM API, and specifying the XSystem's Uniform Resource
    Identifier (XRI) string as its parameter. It is expected that the XRI will
    be specified by the local storage system administrators, and applications
    should strive to make this easily configured at rum time.

    The XAM Library utilizes this method to initiate a connection to
    an (already created) XSystem instance.

    @note The XSystem is not fully usable until it has been authenticated.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inXRI A xam_string. It contains the XSystem's Uniform Resource
                 Identifier. A BNF of this format is listed below:

                      xsystemname[?param=value[{&param=value}]]

                 The xsystemname is vendor specific - it may be an IP address, or
                 some other id. It may not contain '/', '?', or '!' characters.
                 Additionally, param'='value pairs can be specified
    @param inHandle A reference to valid storage for an xsystem_handle.
                    This contains an XSystem handle that was created by a call to
                    VIM_CreateXSystem.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_Connect (const xsystem_handle inHandle,
                     const xam_string inXRI);

/**
    Allows an application to authenticate an XSystem. It provides a generic
    interface to exchange data as part of the authentication process. The
    application should check the XSystem property xyz to determine which
    patterns of authentication (mechanisms) are available for use. After a
    pattern is selected, the appropriate sequence of data exchanges should be
    made (using this call) in order to authenticate. A failed authentication
    will make the XSystem unusable – applications cannot repeat failed
    authentications using the same XSystem.

    @note The outXStream must be closed (using XStream_Close() function) when
          the application has finished its authentication processing.

    @note If the XSystem has been closed, or if the inByteCount is set to a
          size larger than the actual number of bytes of storage available in
          the inBuffer, undefined results may occur (this includes but is not
          limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @param inBuffer Data that is being passed to the authentication mechanism
                    is passed in this array of bytes.
    @param inByteCount The number of significant bytes in the passed in buffer.
    @param outXStream A reference to valid storage for an xstream_handle. On
                      return this will contain the XStream handle that was
                      created, and which contains the systems response to the
                      authentication information. The value that is passed in
                      is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_Authenticate (const xsystem_handle inHandle,
                                   const char* inBuffer,
                                   const xam_int inByteCount,
                                   xstream_handle* outXStream);

/**
    Called to release any resources associated with an XSystem. After calling
    this method, the closed XSystem should not be used.

    @note This call will fail if there are any open XSets associated with this
          XSystem.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_Close (const xsystem_handle inHandle);



/**
    An XSystem in its normal state will generate an error when an application
    attempts to close it if it has open XSets in it. Making this call will
    change the state of the XSystem and allow it to be closed without regard
    for any open XSets. Note that the XSystem will no longer be usable after
    this call is made, and the only call that will succeed is an XSystem.Close.

    @note This is a VERY DANGEROUS call that may result in data loss if used
          inappropriately. It is recommended that applications track all open
          XSets, and close the XSets properly as opposed to making this call.

   @note If the XSystem has been closed undefined results may occur (this
         includes but is not limited to data loss and data corruption).

   Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_Abandon (const xsystem_handle inHandle);

/**
    Evaluates all retention criteria that exists on a given XSet, specified
    as a xam_xuid, and shall return TRUE if there exists retention criteria
    which would prohibit XSet deletion. The method returns FALSE if all XSet
    retention criteria have been met.

   This method does not evaluate the on-hold status.

    A non-fatal error will be returned if the specified XUID is improperly
    formatted, does not exist in the XSystem, or if the caller is not
    authorized to access the XSet.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xsystem_handle.
```

```
        @param inXUID The XUID of the XSet to be checked.
        @param outIsRetained A reference to valid storage for a xam_boolean.
                             On return this will be set to true if the XSet is
                             under retention in accordance with the XSet retention
                             criteria, false otherwise. The value that is
                             passed in is not used, and is overwritten with the
                             result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_IsXSetRetained (const xsystem_handle inHandle,
                                const xam_xuid inXUID,
                                xam_boolean* const outIsRetained);

    /**
        Deletes an XSet from the XSystem.

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be deleted.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_DeleteXSet (const xsystem_handle inHandle,
                                    const xam_xuid inXUID);

    /**
        Places an XSet on hold. A held XSet cannot be changed in any way (e.g. the
        XSet can only be opened in read mode and commits of a held XSet will fail).

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be held.
        @param inHoldID A xam_string that contains the ID to be associated with the
                        hold.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_HoldXSet (const xsystem_handle inHandle,
                                  const xam_xuid inXUID,
                                  const xam_string inHoldID);
    /**
        Releases a specific hold on an XSet (associated with the hold id).
        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be held.
        @param inHoldID A xam_string that contains the ID associated with the hold.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_ReleaseXSet (const xsystem_handle inHandle,
                                      const xam_xuid inXUID,
                                      const xam_string inHoldID);

    /**
        Checks the accessibility of an XSet on the XSystem. It is not an error if
        the XSet does not exist on the XSystem: such an XSet shall be noted as
        being inaccessible.

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be checked.
        @param inMode The bitwise OR of the access 'permissions' to be checked:
                       - XSET_R_OK for read permission
                       - XSET_W_OK for write permission
                       - XSET_D_OK for delete permission
        @param outIsAccessible A reference to valid storage for a xam_boolean.
                               On return this will be set to true if the XSet is
                               accessible according to the access permissions
                               set by mode, false otherwise. The value that is
                               passed in is not used and is overwritten with the
                               result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_AccessXSet (const xsystem_handle inHandle,
                                     const xam_xuid inXUID,
                            const xam_int inMode,
                                     xam_boolean* outIsAccessible);

    /**
        Gets the time at which the XSet was last opened or committed, whichever is
        the most recent.

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).
        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be checked.
        @param outAccessTime A reference to valid storage for a xam_datetime. On
                             return this will be set to the time at which the XSet
                             was last opened or committed, whichever is the most
                             recent. The value that is passed in is not used and
                             is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_GetXSetAccessTime (const xsystem_handle inHandle,
                                   const xam_xuid inXUID,
                                   xam_datetime* outAccessTime);

    /**
        Creates a new empty XSet associated with the XSystem. Note that this XSet
        will not exist on the XSystem unless that XSet is committed.

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inMode A string indicating the mode to open the XSet in.
            Possible values are:
            o “restricted” - open for reading and limited writing. Adding, deleting
                             or modifying fields that are binding is not allowed.
                             Changing fields from binding to nonbinding (or vice
                             versa) is not allowed. Commit of the XSet instance
                             will fail if any binding fields havebeen modified.
                             Successful commit of the XSet will never generate a
                             new XUID.
            o “unrestricted” - open for reading and writing. There are no limits
                               on adding, deleting or modifying fields; nor are
                               there limits on changing fields from binding to
                               nonbinding (or vice versa). Successful commit of the
                               XSet will generate a new XUID if any binding fields
                               have been added, deleted, or modified, or if any
                               fields have been changed from binding to nonbinding
                               (or vice versa).
        @param outXSet A reference to valid storage for an xset_handle. The value
                       that is passed in is not used and is overwritten with the
                       result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSystem_CreateXSet (const xsystem_handle inHandle,
                            const xam_string inMode,
                                      xset_handle* outXSet);

    /**
        Creates or modifies a property field with the name of
        ".xset.access.policy" and a type set to "application/vnd.snia.xam.string"
        on the object referenced by the passed-in xset_handle. Its value and
        binding attributes will be set according to the user-provided parameters.
        This field will be used by the XAM Storage System to determine the policies
        to use when accessing this XSet..

        @note If an access policy has not been applied to an XSet at the time of
              the initial commit, then the property will be created and set as the
              default access policy of the XSystem (i.e. the first string in the
              XSystem AccessPolicyList).

        @note Changing this policy from binding to nonbinding (or the converse)
              will result in a new XSet being created and a new XUID being
              assigned on commit.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param inBinding A xam_boolean set to true if the field should be binding;
                         or false otherwise.
        @param inPolicy A xam_string containing the name of the policy to be
                        applied.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_ApplyAccessPolicy (const xset_handle inHandle,
                                const xam_boolean inBinding,
                                const xam_string inPolicy);

    /**
        Removes all access fields from the XSet.

        @note If an access policy has not been applied to an XSet at the time of
              the initial commit, then the property will be created and set as
              the default access policy of the XSystem (i.e., the first string
              in the XSystem AccessPolicyList).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that contains
                        the access field(s).
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_ResetAccessFields (const xset_handle inHandle);

    /**
```

```
    Creates a property field with the name of "xam.management_policy" and a
    type set to "application/vnd.snia.xam.string" on the object referenced by
    the passed in xam_handle_t. Its value and binding attributes will be set in
    accordance with the user-provided parameters. This field will be used by
    the XAM storage system to determine the default policies to use when
    managing this XSet.

    @note If a management policy has not been applied to an XSet at the time
          of the initial commit, then the property will be created and set as
          the default management policy of the XSystem (i.e. first string in
          the XSystem ManagementPolicyList).

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ApplyManagementPolicy (const xset_handle inHandle,
                                        const xam_boolean inBinding,
                                        const xam_string inPolicy);

/**
    Removes all management fields from the XSet. This includes the
    ".xset.retention.start_time"; because this is a binding field, calling this
    method will always result in a new XUID being assigned to this XSet at the
    next commit.

    @note If a management policy has not been applied to an XSet at the time of
          the initial commit, then the property will be created and set as the
          default management policy of the XSystem (i.e. first string in the
          XSystem ManagementPolicyList).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ResetManagementFields (const xset_handle inHandle);

/**
```

```
    Creates a scope to for storing and evaluating retention criteria. It
    creates a field with a type of “application/vnd.snia.xam.string” and
    sets the value to the retention id. The field name is formed by appending
    the retention id to the following prefix: “.xset.retention.list.”. Thus
    the final format of the name is .xset.retention.list.<retention id>. It
    will have its binding attribute set according to the binding flag
    set by the application.

    @note Changing this field from binding to nonbinding (or vice versa)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_CreateRetention (const xset_handle inHandle,
                          const xam_boolean inBinding,
                          const xam_string inRetentionID);

/**
    Enables or disables retention scoped by the specified retention id.
    This flag is stored in a field of type “application/vnd.snia.xam.boolean”.
    The name of the field is formed by inserting the retention id between a
    prefix (.xset.retention.) and a suffix (.enabled); thus, the final format
    of the name is .xset.retention.<retention id>.enabled. If the field does
    not exist it will be created; otherwise the value will be updated if and
    only if the value is changed from false to true - if the value is set to
    true it cannot be changed. It will have its binding attribute set in
    accordance with the binding flag that is set by the application.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inEnabled A xam_boolean containing a flag indicating if event
                     retention is enabled on this XSet or not. If the flag is
                     set to true, event retention is enabled, otherwise it is
                    disabled.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetRetentionEnabledFlag (const xset_handle inHandle,
                                  const xam_string inRetentionID,
                                  const xam_boolean inBinding,
                                  const xam_boolean inEnabled);

/**
    This method will enabled or disable retention scoped by the specified
    retention id. The policy name of the policy holding the enabled flag
    is stored in a field of type "application/vnd.snia.xam.string". The name
    of the field is formed by inserting the retention id between a prefix
    (.xset.retention.) and a suffix (.enabled.policy); thus, the final format
    of the name is .xset.retention.<retention id>.enabled.policy. If the field
    does not exist it will be created; otherwise the value will be updated if
    and only if the value is changed from false to true - if the value is set
    to true it cannot be changed. It will have its binding attribute set in
    accordance with the binding flag that is set by the application.

    @note If the .xset.retention.<retention id>.enabled field is also present
          on the XSet, it will be used by the XAM Storage System in preference
          to this field.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ApplyRetentionEnabledPolicy (const xset_handle inHandle,
                                      const xam_string inRetentionID,
                                      const xam_boolean inBinding,
                                      const xam_string inPolicy);

/**
    Sets the duration of retention scoped by the specified retention id.
    This flag is stored in a field of type "application/vnd.snia.xam.int".
    The name of the field is formed by inserting the retention id between
    a prefix (.xset.retention.) and a suffix (.duration); thus, the final
    format of the name is .xset.retention.<retention id>.duration. If the
    field does not exist it will be created; otherwise the value will be
    updated if and only if the duration is increased. It will have its
    binding attribute set according to the binding flag that is set by the
    application.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inDuration A xam_int containing the amount of time (measured in
                      milliseconds from the time of commit) to retain the XSet.
                      Zero indicates no retention, while a negative one (-1)
                      indicates infinite retention.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
xam_status
VIM_XSet_SetRetentionDuration (const xset_handle inHandle,
                               const xam_string inRetentionID,
                               const xam_boolean inBinding,
                               const xam_int inDuration);

/**
    Sets the duration of retention scoped by the specified retention id. This
    policy name is stored in a field of type “application/vnd.snia.xam.string”.
    The name of the field is formed by inserting the retention id between a
    prefix (.xset.retention.) and a suffix (.duration.policy); thus, the final
    format of the name is .xset.retention.<retention id>.duration.policy. If
    the field does not exist it will be created; otherwise the value will be
    updated if and only if the duration is increased. It will have its binding
    attribute set according to the binding flag that is set by the application.

    @note If the .xset.retention.<retention id>.duration field is also present
          on the XSet, it will be used by the XAM Storage System in preference
          to this field.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ApplyRetentionDurationPolicy (const xset_handle inHandle,
                                       const xam_string inRetentionID,
                                       const xam_boolean inBinding,
                                       const xam_string inPolicy);
/**
    Sets the start time of retention scoped by the specified retention id. The
    current time of the XSystem is stored in a field of type
    "application/vnd.snia.xam.datetime". The name of the field is formed by
    inserting the retention id between a prefix (.xset.retention.) and a suffix
    (.starttime); thus, the final format of the name is
    .xset.retention.<retention id>.starttime. If the field does not exist, it
    will be created; otherwise, an error will be generated, as it is not allowed
    to change the starttimme once set. It will have its binding attribute
    set according to the binding flag that is set by the application.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being created.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetRetentionStarttime (const xset_handle inHandle,
                                const xam_string inRetentionID,
                                const xam_boolean inBinding);

/**
    If this XSet does not already contain the field .xset.retention.list.base,
    this method will create the field with a type of
    "application/vnd.snia.xam.string" and set the value to "base". It will also
    create the "application/vnd.snia.xam.boolean" field
    .xset.retention.base.enabled and set the value to true. The duration will
    be stored in a field named .xset.retention.base.duration. This field is of
    type "application/vnd.snia.xam.int". If the field already exists, its value
    will be changed to match the passed in duration if and only if the duration
    of the retention is not reduced; the method will generate an error if the
    duration is reduced. If the field does not already exist, it will be created
    with the specified duration as the value. These fields will have their
    binding attribute set according to the binding flag that is set by the
    application. These fields will be used by the XAM Storage System to
    determine the base retention duration to use when managing this XSet.
    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    @note When an XSet instance containing the field .xset.retention.list.base
          is first committed, the field .xset.retention.base.starttime will be
          created and have its value set to .xset.time.xuid.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inDuration A xam_int containing the amount of time (measured in
                      milliseconds from the time of commit) to retain the XSet.
                      Zero indicates no retention, while a negative one (-1)
                      indicates infinite retention.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
XSet_SetBaseRetention (const xset_handle inHandle,
                       const xam_boolean inBinding,
                       const xam_int inDuration);

/**
    If this XSet does not already contain the field .xset.retention.list.base,
    this method will create the field with a type of
    "application/vnd.snia.xam.string" and set the value to "base". It will also
    create the "application/vnd.snia.xam.boolean" field
    .xset.retention.base.enabled and set the value to true. The duration policy
    will be stored in a field named .xset.retention.base.duration.policy. This
    field is of type "application/vnd.snia.xam.string". If the field already
    exists, its value will be changed to match the passed in policy if and only
    if the policy would not reduce the duration of the retention; the method
    will generate an error if the policy reduces the duration. If the field
    does not already exist, it will be created with the specified policy name
    as the value. These fields will have their binding attribute set in
    accordance with the binding flag that is set by the application. These fields
                                   will
    be used by the XAM Storage System to determine the base retention duration
    to use when managing this XSet.

    @note If the .xset.retention.base.duration field is also present on the
          XSet, it will be used by the XAM Storage System in preference to
          this policy field.

    @note When an XSet instance containing the field .xset.retention.list.base
          is first committed, the field .xset.retention.base.starttime will be
          created and have its value set to .xset.time.xuid.

    @note Changing this field from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
xam_status
XSet_ApplyBaseRetentionPolicy (const xset_handle inHandle,
                               const xam_boolean inBinding,
                               const xam_string inPolicy);

/**
    Creates a property field on the specified XSet with the name of
    "xam.autodelete_policy" and a type set to "application/vnd.snia.xam.string"
    Its value and binding attributes will be set according to the user
    provided parameters. This field will be used by the XAM storage system to
    determine if the XSet should be automatically deleted upon expiration of
    retention. Applying the policy will also remove the "xam.autodelete" from
    the XSet.

    @note If the explicit duration field is also present on the XSet
          ("xam.autodelete") it will be used by the XAM storage system in
          preference to this field.

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ApplyAutoDeletePolicy (const xset_handle inHandle,
                                         const xam_boolean inBinding,
                                         const xam_string inPolicy);

/**
    Creates a property field on the specified XSet with the name of
    "xam.autodelete" and a type set to "application/vnd.snia.xam.boolean". Its
    value and binding attributes will be set according to the user
    provided parameters. This field will be used by the XAM storage system to
    determine if the XSet should be automatically deleted upon expiration of
    retention. Applying the policy will also remove the "xam.autodelete_policy"
    field from the XSet.

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being assigned
          on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inAutoDelete A xam_boolean containing a flag indicating if
                        autodelete is enabled on this XSet or not. If the flag
                        is set to true, autodelete is enabled, otherwise it is
                        disabled.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetAutoDelete (const xset_handle inHandle,
                                  const xam_boolean inBinding,
                                  const xam_boolean inAutoDelete);


/**
    If this XSet does not have an auto shred policy applied to it, this method
    will create a property field on the specified XSet with the name of
    ".xset.deletion.shred.policy" and a type set to
    "application/vnd.snia.xam.string". Its value and binding attributes will be
    set according to the user-provided parameters. If the field already exists
    on the XSet, then its value will be updated with the specified value.
    This field will be used by the XAM Storage System to determine if the XSet
    should be shredded after XSet deletion. If the ".xset.deletion.shred" field
    is also present on the XSet it will be used by the XAM Storage System in
    preference to this field.

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
```

```
VIM_XSet_ApplyShredPolicy (const xset_handle inHandle,
                           const xam_boolean inBinding,
                           const xam_string inPolicy);

/**
    If this XSet does not have auto shred set on it, this method will create a
   property field on the specified XSet with the name of ".xset.deletion.shred" and
                                 a
    type set to "application/vnd.snia.xam.boolean". Its value and binding
    attributes will be set according to the user-provided parameters. If the
    field already exists on the XSet, then its value will be updated with the
    specified value. This field will be used by the XAM Storage System to
    determine if the XSet should be shredded after deletion.

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being assigned
          on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inShred A xam_boolean containing a flag indicating if shredding is
                   enabled on this XSet or not. If the flag is set to TRUE,
                   shredding is enabled, otherwise it is disabled.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_SetShred (const xset_handle inHandle,
                   const xam_boolean inBinding,
                   const xam_boolean inShred);

/**
    Creates a property field on the specified XSet with the name of
    "xam.storage_policy" and a type set to "application/vnd.snia.xam.string".
    Its value and binding attributes will be set according to the user
    provided parameters. This field will be used by the XAM storage system to
    determine the storage policy of the XSet.

    @note Changing this policy from binding to nonbinding (or the converse)
          will result in a new XSet being created and a new XUID being
          assigned on commit.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inBinding A xam_boolean set to true if the field should be binding;
                     or false otherwise.
    @param inPolicy A xam_string containing the name of the policy to be
                    applied.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_ApplyStoragePolicy (const xset_handle inHandle,
                                      const xam_boolean inBinding,
                                      const xam_string inPolicy);

/**
    Evaluates all factors that affect the retention duration that is
    currently in effect for the given retention id, and returns that
    duration to the caller.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param inRetentionID A xam_string containing the retention identifier of the
                         retention being checked.
    @param outDuration A reference to valid storage for a xam_int. On return
                       this will be set to the actual minimum retention duration
                       that is currently being in effect for the XSet after
                       evaluating the policies. The value that is passed in is
                       not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetActualRetentionDuration (const xset_handle inHandle,
                                     const xam_string inRetentionID,
                                     xam_int* outDuration);

/**
    Evaluates all factors that affect if this retention is enabled for the
    XSet, and return that enabled state to the caller.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle A valid xset_handle. This is the object that will contain
                    the new field.
    @param outEnabled A reference to valid storage for a xam_boolean. On return
                      this will be set to match the enabled state in effect for
                      the XSet after evaluating the policies. The value that is
                      passed in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_GetActualRetentionEnabled (const xset_handle inHandle,
                                    const xam_string inRetentionID,
                                    xam_boolean* outEnabled);
/**
    Evaluates all factors that affect if auto delete is enabled for the XSet,
    and return that enabled state to the caller.
```

```
        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param outEnabled A reference to valid storage for a xam_boolean. On return
                          this will be set to match the enabled state in effect for
                          the XSet after evaluating the policies. The value that is
                          passed in is not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_GetActualAutoDelete (const xset_handle inHandle,
                                             xam_boolean* outEnabled);

    /**
        Evaluates all factors that affect if shredding is enabled for the XSet and
        return that enabled state to the caller.

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle A valid xset_handle. This is the object that will contain
                        the new field.
        @param outEnabled A reference to valid storage for a xam_boolean. On return
                          this will be set to match the enabled state in effect for
                          the XSet after evaluating the policies. The value that is
                          passed in is not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_GetActualShred (const xset_handle inHandle,
                             xam_boolean* outEnabled);

    /**************************************************************************
     *  method prototypes for editing an XSet
     **************************************************************************/

    /**
        Opens an XSet in the XSystem.

        @note If the XSystem has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xsystem_handle.
        @param inXUID The XUID of the XSet to be opened.
        @param inMode A string indicating the mode to open the XSet in:
            o "readonly" - open for reading. Adding, deleting or modifying fields
```

```
                      is not allowed. Commit of the XSet instance will fail.
        o “restricted” - open for reading and limited writing. Adding, deleting
                         or modifying fields that are binding is not allowed.
                         Changing fields from binding to nonbinding (or vice
                         versa) is not allowed. Commit of the XSet instance
                         will fail if any binding fields havebeen modified.
                         Successful commit of the XSet will never generate a
                         new XUID.
        o “unrestricted” - open for reading and writing. There are no limits
                           on adding, deleting or modifying fields; nor are
                           there limits on changing fields from binding to
                           nonbinding (or vice versa). Successful commit of the
                           XSet will generate a new XUID if any binding fields
                           have been added, deleted, or modified, or if any
                           fields have been changed from binding to nonbinding
                           (or vice versa).
        o “copy” – open for reading and writing. There are no limits on adding,
                   deleting or modifying fields; nor are there limits on
                   changing fields from binding to nonbinding (or vice versa).
                   The first successful commit will always generate a new XUID.
                   Subsequent successful commits of the XSet will generate a
                   new XUID if any binding fields have been added, deleted, or
                   modified, or if any fields have been changed from binding to
                   nonbinding (or vice versa).
    @param outXSet A reference to valid storage for a xset_handle. On return
                   this will contain the XSet handle. The value that is passed
                   in is not used and is overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_OpenXSet (const xsystem_handle inHandle,
                                const xam_xuid inXUID,
                      const xam_string inMode,
                                xset_handle* outXSet);

/**
    Creates a copy of an XSet in the XSystem, returning a handle to an
    XSet instance associated with the XSystem. This XSet will not exist
    on the XSystem unless that XSet instance is committed.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete. For applications that wish
        to use a non-blocking version of this method, refer to
        "XSystem_AsyncCopyXSet".

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to open the XSet in:
        o “restricted” - open for reading and limited writing. Adding, deleting
                         or modifying fields that are binding is not allowed.
                         Changing fields from binding to nonbinding (or vice
                         versa) is not allowed. Commit of the XSet instance
                         will fail if any binding fields havebeen modified.
                         Successful commit of the XSet will never generate a
```

```
                        new XUID.
          o “unrestricted” - open for reading and writing. There are no limits
                            on adding, deleting or modifying fields; nor are
                            there limits on changing fields from binding to
                            nonbinding (or vice versa). Successful commit of the
                            XSet will generate a new XUID if any binding fields
                            have been added, deleted, or modified, or if any
                            fields have been changed from binding to nonbinding
                            (or vice versa).
      @param outXSet A reference to valid storage for a xset_handle. On return
                     this will contain the XSet handle. The value that is passed
                     in is not used and is overwritten with the result.
      @return The status code generated by calling this function. Use the
              XAM_GetErrorToken function to determine the meaning of this value.
   */
  EXPORT xam_status DECL
  VIM_XSystem_CopyXSet (const xsystem_handle inHandle,
                        const xam_xuid inXUID,
                        const xam_string inMode,
                        xset_handle* outXSet);

  /**
      Stores an XSet in the XSystem. Note this does not close the XSet, which can
      still be modified as allowed by the authorization of the XSystem. A XUID
      will be assigned by the XAM storage system and this XUID will be returned.

      Open XStreams will not cause the commit to fail. Only the data that was
      successfully written to such XSteams will be committed.

      If this is a modified XSet (e.g. an existing XSet was opened and changed)
      then a new XUID may or may not be assigned by the commit, in accordance
      with the following rules:
          - If only variable fields are edited (created, deleted, or changed)
            then the XAM storage system may not assign a new XUID.
          - If any binding fields are edited (created, deleted, or changed) then
            the XAM storage system must assign a new XUI.

     In any case, an application should be coded to handle cases where the XUID
     changes when a modified XSet is committed.

     If a management policy has not been applied to the XSet prior to commit, a
     default management policy will be applied to the XSet at the time of commit.

      @note If the XSystem has been closed undefined results may occur (this
            includes but is not limited to data loss and data corruption).

      Concurrency requirements:
          This method is thread-safe.
      Blocking:
          This method will block until complete.

      @param inHandle An xset_handle.
      @param outXUID A reference to valid storage for a XUID. On return this will
                     contain the XUID that was assigned to the XSet by the XAM
                     storage system. The value that is passed in is not used and
                     is overwritten with the result.
      @return The status code generated by calling this function. Use the
              XAM_GetErrorToken function to determine the meaning of this value.
   */
  EXPORT xam_status DECL
```

```
VIM_XSet_Commit (const xset_handle inHandle,
                            xam_xuid* outXUID);

/**
    Releases any resources associated with an XSet. After calling this method,
    the closed XSet should not be used.

    @note This call will fail if there are any open XStreams associated with
          this XSet.

    @note if the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xset_handle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_Close (const xset_handle inHandle);

/**
    An XSet in its normal state will generate an error when an application
    attempts to close it if there are open XStreams in it. Making this call
    will change the state of the XSet and allow it to be closed without regard
    for any open XStreams. Note that the XSet will no longer be usable after
    this call is made, and the only call that will succeed is an XSet.Close.

    @note this is a VERY DANGEROUS call that may result in data loss if used
          inappropriately. It is recommended that applications track all open
          XStreams, and close the XStreams properly as opposed to making this
          call.

    @note If the XSet has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xset_handle
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_Abandon (const xset_handle inHandle);

/***********************************************************************
 *  method prototypes for XSet migration
 ***********************************************************************/

/**
    Opens an export XStream for the XSet. The XSet must have been committed,
    and must not have been modified since it was opened / committed. The XSet
    will enter an import/export state, and will thus generate errors if used
```

```
        for any operation until the export XStream is closed. The original XSet
        referred to by the XSet handle will be overwritten.

        The XStream will contain a canonical representation of the XSet. This data
        can be read from the XStream using normal XStream calls and semantics. When
        the XStream is closed the XSet will return to a normal state.


        @note If the XSet has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xset_handle.
        @param outXStream A reference to valid storage for a xstream_handle. On
                          return this will contain the XStream handle of an XStream
                          opened in "readonly" mode. The value that is passed in is
                          not used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_OpenExportXStream (const xset_handle inHandle,
                                       xstream_handle* outXStream);

    /**
        Opens an import XStream for the XSet. The XSet will enter an import/export
        state, and will thus generate errors if used for any operation until the
        XStream is closed. Any data in the original XSet instance will be
        overwritten.

        It is expected that a data stream containing the canonical representation
        of an XSet will be written into the XStream. When the XStream is closed,
        the data will be validated. If the data is determined to be valid, then the
        XSet will return to a normal state (i.e. will no longer generate errors
        when operated on) but it will now refer to the XSet that was described by
        the canonical data that was written to the XStream. If the validation of the
        data fails (i.e. it contains invalid or improperly formatted data) then the
        XSet will enter a corrupted state. It will no longer be recoverable and all
        operations except abandon (followed by close) will fail.

        After a successful validation, the XSet fields can be examined as any
        normal fields. The XSet can be modified. The XSet is not committed, but it
        is in all ways a normal XSet, and may be committed as per normal XSet
        semantics. If the XSet is committed prior to any modification to binding
        fields (adding, modifying or deleting binding fields; or changing the
        binding attribute of any fields) then the XUID will be the XUID described
        by the import XStream. Modification to any binding fields as decribed above
        will result in a new XUID being assigned upon commit.

        @note If the XSet has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.
        @param inHandle an xset_handle.
        @param outXStream A reference to valid storage for a xstream_handle. On
                          return this will contain the XStream handle of an XStream
                          opened in “w” mode. The value that is passed in is not
                          used and is overwritten with the result.
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_OpenImportXStream (const xset_handle inHandle,
                                           xstream_handle* outXStream);

    /**********************************************************************
     *  method prototypes for job control
     **********************************************************************/

    /**
        Submits a job request to the XAM storage system. Fields on the XSet will be
        evaluated as input to the job according to the semantics of the XAM
        job control subsystem (refer to the XAM architecture document for more
        details). This XSet will be used to communicate health and status
        information about the job, as well as any results from the job.

        @note If the XSet has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xset_handle
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_SubmitJob (const xset_handle inHandle);

    /**
        Stops a currently running job in XAM storage system, if the XSet was used
        to start a job. Fields on the XSet will be evaluated as input to the job in
        accordance with the semantics of the XAM job control subsystem (refer to
        the XAM architecture document for more details).

        @note If the XSet has been closed undefined results may occur (this
              includes but is not limited to data loss and data corruption).

        Concurrency requirements:
            This method is thread-safe.
        Blocking:
            This method will block until complete.

        @param inHandle An xset_handle
        @return The status code generated by calling this function. Use the
                XAM_GetErrorToken function to determine the meaning of this value.
     */
    EXPORT xam_status DECL
    VIM_XSet_HaltJob (const xset_handle inHandle);
```

```
/**********************************************************************
 *  method prototypes for asynchronous i/o
 **********************************************************************/

/**
    Asynchronously opens an XSet in the XSystem.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to open the XSet in:
       o “readonly” - open for reading. Adding, deleting or modifying fields
                      is not allowed. Commit of the XSet instance will fail.
       o “restricted” - open for reading and limited writing. Adding, deleting
                        or modifying fields that are binding is not allowed.
                        Changing fields from binding to nonbinding (or vice
                        versa) is not allowed. Commit of the XSet instance
                        will fail if any binding fields havebeen modified.
                        Successful commit of the XSet will never generate a
                        new XUID.
       o “unrestricted” - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).

    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_AsyncOpenXSet (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           const xam_string inMode,
                           const XOPID inXOPID,
                           xasync_callback inCallback,
                           xasync_handle* outAsyncHandle);

/**
```

```
    Begins the asynchronous copying of an XSet in the XSystem, ultimately
    returning a handle to an XSet instance associated with the XSystem.
    The specified callback will be invoked as part of the asynchronous
    copying. To monitor the status of this operation, the application can
    poll the Async instance that is generated by this method. A handle to
    an XAsync instance is also passed to any provided callback method when
    that callback method is invoked.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xsystem_handle.
    @param inXUID The XUID of the XSet to be opened.
    @param inMode A string indicating the mode to copy the XSet in:
       o “restricted” - open for reading and limited writing. Adding, deleting
                        or modifying fields that are binding is not allowed.
                        Changing fields from binding to nonbinding (or vice
                        versa) is not allowed. Commit of the XSet instance
                        will fail if any binding fields havebeen modified.
                        Successful commit of the XSet will never generate a
                        new XUID.
       o “unrestricted” - open for reading and writing. There are no limits
                          on adding, deleting or modifying fields; nor are
                          there limits on changing fields from binding to
                          nonbinding (or vice versa). Successful commit of the
                          XSet will generate a new XUID if any binding fields
                          have been added, deleted, or modified, or if any
                          fields have been changed from binding to nonbinding
                          (or vice versa).
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_AsyncCopyXSet (const xsystem_handle inHandle,
                           const xam_xuid inXUID,
                           const xam_string inMode,
                           const XOPID inXOPID,
                           xasync_callback inCallback,
                           xasync_handle* outAsyncHandle);

/**
    Asynchronously creates an open XStream instance in either “readonly”
    or “writeonly” mode, based on the mode argument.
    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle A valid XSystem handle which contains the XStream field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance.
        o “appendonly”: open for writing. Read and seek methods will fail on the
                        XStream instance. Appends the existing data in the XStream.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSystem_AsyncOpenXStream (const xsystem_handle inHandle,
                              const xam_string inName,
                              const xam_string inMode,
                              const XOPID inXOPID,
                              xasync_callback inCallback,
                              xasync_handle* outAsyncHandle);

/**
    Asynchronously creates an open XStream instance in either “readonly”
    or “writeonly” mode, based on the mode argument.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle A valid XSet handle which contains the XStream field.
    @param inName A xam_string containing the name of the field to be created.
    @param inMode A string indicating the mode to open the XStream in:
        o “readonly”: open for reading. Write methods will fail on the XStream
                      instance.
        o “writeonly”: open for writing. Read and seek methods will fail on the
                       XStream instance.
        o “appendonly”: open for writing. Read and seek methods will fail on the
                        XStream instance. Appends the existing data in the XStream.
    @param inXOPID Unique ID that is specified by the application to identify
```

```
                 the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_AsyncOpenXStream (const xset_handle inHandle,
                           const xam_string inName,
                           const xam_string inMode,
                           const XOPID inXOPID,
                           xasync_callback inCallback,
                           xasync_handle* outAsyncHandle);

/**
    Asynchronously transfers data from the storage system into the target
    buffer, up to the number of bytes requested.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    @note If the inBufferLength is set to a size larger than the actual
          number of bytes of storage available in the inBuffer, undefined
          results may occur (this includes but is not limited to data loss and
          data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle that must have been opened in read mode.
    @param ioBuffer A byte array to read the data into.
    @param inBufferLength A xam_int set to the number of bytes in the buffer.
    @param inXOPID Unique ID that is specified by the application to identify
                 the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_AsyncRead (const xstream_handle inHandle,
                       char* ioBuffer,
                       const xam_int inBufferLength,
                       const XOPID inXOPID,
                       xasync_callback inCallback,
                       xasync_handle* outAsyncHandle);

/**
    Asynchronously transfers data from the source buffer to the XAM storage
    system, up to the number of bytes requested.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    @note This method may fail with an error if the maximum number of bytes
          supported in an XStream is reached. All XSystems must support at
          least XXX bytes in an XStream. However, some XAM storage systems may
          support more than this. To determine the actual maximum number of
          bytes allowed in an XStream an application should evaluate the YYY
          field on the XSystem. For more information on this topic please
          consult the XAM architecture document.

    @note If the inByteCount is set to a size larger than the actual number of
          bytes of storage available in the inBuffer, undefined results may
          occur (this includes but is not limited to data loss and data
          corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle that must have been opened in write mode.
    @param inBuffer A byte array containing the data to be written.
    @param inByteCount A xam_int set to the number of bytes in the buffer to be
                       written.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_AsyncWrite (const xstream_handle inHandle,
                        const char* inBuffer,
                        const xam_int inByteCount,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                        xasync_handle* outAsyncHandle);

/**
    Asynchronously stores an XSet in the XSystem. Note this does not close
    the XSet, which can still be modified as allowed by the authorization
    of the XSystem. A XUID will be assigned by the XAM storage system and this
    XUID will be returned.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    Open XStreams will not cause the commit to fail. Only the data that was
    successfully written to such XSteams will be committed.

    If this is a modified XSet (e.g. an existing XSet was opened and changed)
    then a new XUID may or may not be assigned by the commit, in accordance
    with the following rules:
        - If only variable fields are edited (created, deleted, or changed)
          then the XAM storage system may not assign a new XUID.
        - If any binding fields are edited (created, deleted, or changed) then
          the XAM storage system must assign a new XUI.

   In any case, an application should be coded to handle cases where the XUID
   changes when a modified XSet is committed.

   If a management policy has not been applied to the XSet prior to commit, a
   default management policy will be applied to the XSet at the time of commit.

    @note If the XSystem has been closed undefined results may occur (this
          includes but is not limited to data loss and data corruption).

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xset_handle.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XSet_AsyncCommit (const xset_handle inHandle,
                      const XOPID inXOPID,
                      xasync_callback inCallback,
                      xasync_handle* outAsyncHandle);

/**
    Asynchronously closes a previously opened XStream.
    Any resources that were allocated can be released at this point.

    The VIM is tasked with starting the asynchronous operation and
    immediately returning a valid XAsync handle corresponding to the
    given XOPID. When the operation is complete, the VIM must notify
    the user of it's completion by invoking the provided callback.

    @note Closing an already closed XStream can produce undefined results (this
          includes but is not limited to data loss and data corruption)

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will return immediately.

    @param inHandle An xstream_handle.
    @param inXOPID Unique ID that is specified by the application to identify
                   the asynchronous operation.
    @param inCallback A pointer to a function that is called when the
                      asynchronous operation completes. The parameter passed to
                      the call back function can be probed for information.
    @param outAsyncHandle A handle to the asynchronous operation.
    @return The status code generated by calling this function. Use the
```

```
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XStream_AsyncClose (const xstream_handle inHandle,
                        const XOPID inXOPID,
                        xasync_callback inCallback,
                        xasync_handle* outAsyncHandle);

/*************************************************************************
 *  method prototypes for managing asyncronous operations
 *************************************************************************/

/**
    Stops the operation associated with the passed inHandle

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_Halt (const xasync_handle inHandle);

/**
    Allows the caller to discover if the asynchronous operation relating to the
    passed inHandle is complete or not.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions
    @param outIsComplete A reference to valid storage for a xam_boolean. The
                         result is true if the async operation related to the
                         passed inHandle is complete,
                         or false otherwise.
                         The value that is passed in is not used and is
                         overwritten with the result.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_IsComplete (const xasync_handle inHandle,
                       xam_boolean* outIsComplete);


/**
    Gets the status of the completed asynchronous operation that relates
    to the passed inHandle.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.
    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions
    @param outStatus A reference to valid storage for a xam_status.
          On input this param is not used, on output this param is populated
          with the status of the completed asynchronous operation that relates
          to the passed inHandle.

          If the underlying asynchronous operation is not complete
          this function will fail and return a status for this call which
          relates to the failure.

    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetStatus (const xasync_handle inHandle,
                      xam_status* outStatus);

/**
    Gets the XOPID that was set by the application for the asynchronous
    operation that relates to the passed inHandle

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXOPID A reference to valid storage for a XOPID.
                    On input this param is not used.
                    On output (if function is successful) this param is
                              populated with the XOPID of the asynchronous
                              operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetXOPID (const xasync_handle inHandle,
                     XOPID* outXOPID);


/**
    Gets the XSet of the completed asynchronous operation that relates to the
    passed inHandle. The return status of this function is set appropriately on
    success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.
    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXSet A reference to valid storage for an xset_handle.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the XSet of the asynchronous
                             operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetXSet (const xasync_handle inHandle,
                    xset_handle* outXSet);


/**
    Gets the XStream from the completed asynchronous operation that relates to the
    passed inHandle. The return status of this function is set appropriately on
    success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXStream A reference to valid storage for an xstream_handle.
                      On input this param is not used,
                      On output (if function is successful) this param is
                                populated with the XStream from the asynchronous
                                operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetXStream (const xasync_handle inHandle,
                       xstream_handle* outXStream);

/**
    Gets the value from a property field with a type set to
    “application/vnd.snia.xam.xuid” on the object referenced by the passed
    inHandle.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outXUID A reference to valid storage for a xam_xuid.
                   On input this param is not used,
                   On output (if function is successful) this param is
                             populated with the xam_xuid of the asynchronous
                             operation that relates to the passed inHandle.
    @return The status code generated by calling this function. Use the
```

```
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetXUID (const xasync_handle inHandle,
                    xam_xuid* outXUID);


/**
    Gets the number of bytes read from the completed asynchronous operation
    that relates to the passed inHandle. The return status of this function
    is set appropriately on success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous read or this function will not be successful.

    @note The asynchronous operation that relates to the passed inHandle must
          be completed for this function call to be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outBytesRead A reference to valid storage for a xam_int.
                   On input this param is not used,
                   On output (if function is successful) this param is
                            populated with the number of bytes read
                            during the asynchronous operation that relates to
                            the passed inHandle.
    @return The status code generated by calling this function. Use the
           XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetBytesRead (const xasync_handle inHandle,
                         xam_int* outBytesRead);

/**
    Gets the number of bytes written for the completed asynchronous operation
    that relates to the passed inHandle. The return status of this function
    is set appropriately on success of failure of this call.

    @note The passed inHandle must relate to an operation that performed an
          asynchronous write or this function will not be successful.

    @note The asynchronous operation that relates to the passed inHandle must
          be completed for this function call to be successful.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @param outBytesWritten A reference to valid storage for a xam_int.
                   On input this param is not used,
                   On output (if function is successful) this param is
                            populated with the number of bytes written
```

```
                             during the asynchronous operation that relates to
                             the passed inHandle.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_GetBytesWritten (const xasync_handle inHandle,
                            xam_int* outBytesWritten);

/**
    Releases resources associated with the completed asynchronous operation
    that relates to the passed inHandle.

    Concurrency requirements:
        This method is thread-safe.
    Blocking:
        This method will block until complete.

    @param inHandle An xasync_handle as retrieved by calling anyone of the
                    XXX_AsynchXXX functions.
    @return The status code generated by calling this function. Use the
            XAM_GetErrorToken function to determine the meaning of this value.
 */
EXPORT xam_status DECL
VIM_XAsync_Close (const xasync_handle inHandle);

#ifdef __cplusplus
} // extern "C"
#endif

#endif // __VIM_H_
```

Annex C
(normative)
C API Toolkit

This annex defines toolkit methods that will extend the XAM C API. The goal of the toolkit methods is to make the API easier to use. The methods shall not be incorporated into the same library as the XAM C API, but shall instead be an additional library that coexists with the XAM C API. These toolkit methods shall be implemented in a way that makes no assumptions about any particular implementation of a XAM Library, and shall only link to the public C API methods, never to the private (VIM) methods.

C.1 Field methods

The methods in this section provide convenience functionality for processing fields.

C.1.1 XAMToolkit_IsPropertyField

Syntax prototype:

```
xam_status
XAMToolkit_IsPropertyField (const xam_handle_t inHandle,
                            const xam_string inName,
                            xam_boolean* const outIsProperty);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM Library object reference.
- inName is a xam_string containing the name of the field.
- outIsProperty is a reference to valid storage for a xam_boolean. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The third argument is NULL.
- The field does not exist.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method evaluates the field type and determines if it is a property. If it is a property, then the method will set the passed in value to TRUE; otherwise it will be set to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

C.1.2 XAMToolkit_IsXStreamField

Syntax prototype:

```
xam_status
XAMToolkit_IsXStreamField (const xam_handle_t inHandle,
                           const xam_string inName,
                           xam_boolean* const outIsXStream);
```

Parameters:

- inHandle is a valid xam_handle_t containing an XSet, XSystem, or XAM Library object reference.
- inName is a xam_string containing the name of the field.
- outIsProperty is a reference to valid storage for a xam_boolean. The value that is passed in is not used and is overwritten with the result.

Error conditions:

- The first argument is not a valid xset_handle.
- The second argument is not a valid name (invalid UTF-8).
- The third argument is NULL.
- The field does not exist.
- The xam_handle_t contains an XSet and the XSet has an open import or export stream.
- The xam_handle_t contains an XSet and the XSet is in a corrupt state.
- The xam_handle_t contains an XSet and the XSet is in an abandoned state.
- The xam_handle_t contains an XSystem and the XSystem is in a corrupt state.
- The xam_handle_t contains an XSystem and the XSystem is in an abandoned state.

Description:

This method evaluates the field type and determines if it is an XStream. If it is an XStream, then the method will set the passed in value to TRUE; otherwise it will be set to FALSE.

Concurrency requirements:

This method is thread safe.

Blocking:

This method will block until complete.

C.2 Base64 conversion

To store XUID values in printable formats, it is recommended that applications base64 encode them.

C.2.1 base64_encode

Syntax prototype:

```
void
base64_encode (const char *inSrcBuf, int inSrcLen, char *outDstBuf, int *outDstLen);
```

Parameters:

- inSrcBuf is a pointer to a character string to be encoded in base64.
- inSrcLen is the length of the input character string.
- outDstBuf is a pointer to a buffer where the base64-encoded output is to be placed.
- outDstLen is the length of the base64-encoded output

  Note: To avoid overwriting other data, make sure that outDstBuf is at least (inSrcLen+2)/3 * 4 bytes long.

Error conditions:

None.

C.2.2 base64_decode

Syntax prototype:

```
void
base64_decode (const char *inSrcBuf, int inSrcLen, char *outDstBuf, int *outDstLen);
```

Parameters:

- inSrcBuf is a pointer to a character string to be decoded from base64.
- inSrcLen is the length of the input character string.
- outDstBuf is a pointer to a buffer where the decoded output is to be placed.
- outDstLen is the length of the decoded output

  Note: To avoid overwriting other data, make sure that outDstBuf is at least (inSrcLen+3)/4 * 3 bytes long.

Error conditions:

None.

Annex D (informative) C API Method Mapping

Table D.1, "C Method Name Mapping to XAM Architecture Specification" lists the methods in [XAM-ARCH] and the corresponding method name for the C binding.

Table D.1 – C Method Name Mapping to XAM Architecture Specification

Type	Methods in Arch Spec	Methods in C API Spec
Error token	N/A	XAM_GetErrorToken
Iteration	<XAMHandle>.openFieldIterator	XAM_OpenFieldIterator
	XIterator.next	XIterator_Next
	Xiterator.hasNext	XIterator_HasNext
	XIterator.close	XIterator_Close
Generic Field	<XAMHandle>.containsField	XAM_ContainsField
	<XAMHandle>.setFieldAsBinding	XAM_SetFieldAsBinding
	<XAMHandle>.setFieldAsNonBinding	XAM_SetFieldAsNonbinding
	<XAMHandle>.getFieldType	XAM_GetFieldType
	<XAMHandle>.getFieldLength	XAM_GetFieldLength
	<XAMHandle>.getFieldBinding	XAM_GetFieldBinding
	<XAMHandle>.getFieldReadOnly	XAM_GetFieldReadOnly
	<XAMHandle>.deleteField	XAM_DeleteField
Property Field	<XAMHandle>.createBoolean	XAM_CreateBoolean
	<XAMHandle>.createInt	XAM_CreateInt
	<XAMHandle>.createDouble	XAM_CreateDouble
	<XAMHandle>.createXUID	XAM_CreateXUID
	<XAMHandle>.createString	XAM_CreateString
	<XAMHandle>.createDatetime	XAM_CreateDatetime
	<XAMHandle>.setBoolean	XAM_SetBoolean
	<XAMHandle>.setInt	XAM_SetInt
	<XAMHandle>.setDouble	XAM_SetDouble
	<XAMHandle>.setXUID	XAM_SetXUID
	<XAMHandle>.setString	XAM_SetString
	<XAMHandle>.setDatetime	XAM_SetDatetime
	<XAMHandle>.getBoolean	XAM_GetBoolean
	<XAMHandle>.getInt	XAM_GetInt

Table D.1 – C Method Name Mapping to XAM Architecture Specification

Type	Methods in Arch Spec	Methods in C API Spec
Property Field (cont.)	<XAMHandle>.getDouble	XAM_GetDouble
	<XAMHandle>.getXUID	XAM_GetXUID
	<XAMHandle>.getString	XAM_GetString
	<XAMHandle>.getDatetime	XAM_GetDatetime
XStream Field	<XAMHandle>.createXStream	XAM_CreateXStream
	<XAMHandle>.openXStream	XAM_OpenXStream
	XStream.read	XStream_Read
	XStream.write	XStream_Write
	XStream.seek	XStream_Seek
	XStream.tell	XStream_Tell
	XStream.abandon	XStream_Abandon
	XStream.close	XStream_Close
Connection Administration	XAMLibrary.connect	XAMLibrary_Connect
	XSystem.authenticate	XSystem_Authenticate
	XSystem.close	XSystem_Close
	XSystem.abandon	XSystem_Abandon
XSet Instance	XSystem.createXSet	XSystem_CreateXSet
	XSystem.openXSet	XSystem_OpenXSet
	XSystem.copyXSet	XSystem_CopyXSet
	XSet.commit	XSet_Commit
	XSet.close	XSet_Close
	XSet.abandon	XSet_Abandon
XSet Admin	XSystem.deleteXSet	XSystem_DeleteXSet
	XSystem.holdXSet	XSystem_HoldXSet
	XSystem.releaseXSet	XSystem_ReleaseXSet
	XSystem.accessXSet	XSystem_AccessXSet
	XSystem.getXSetAccessTime	XSystem_GetXSetAccessTime
	XSystem.isXSetRetained	XSystem_IsXSetRetained
Management	XSet.applyAccessPolicy	XSet_ApplyAccessPolicy
	XSet.resetAccessFields	XSet_ResetAccessFields
	XSet.applyManagementPolicy	XSet_ApplyManagementPolicy
	XSet.resetManagementFields	XSet_ResetManagementFields
	XSet.createRetention	XSet_CreateRetention

Table D.1 – C Method Name Mapping to XAM Architecture Specification

Type	Methods in Arch Spec	Methods in C API Spec
Management (cont.)	XSet.setRetentionEnabledFlag	XSet_SetRetentionEnabledFlag
	XSet.applyRetentionEnabledPolicy	XSet_ApplyRetentionEnabledPolicy
	XSet.setRetentionDuration	XSet_SetRetentionDuration
	XSet.applyRetentionDurationPolicy	XSet_ApplyRetentionDurationPolicy
	XSet.setRetentionStarttime	XSet_SetRetentionStarttime
	XSet.setBaseRetention	XSet_SetBaseRetention
	XSet.applyBaseRetentionPolicy	XSet_ApplyBaseRetentionPolicy
	XSet.applyAutoDeletePolicy	XSet_ApplyAutoDeletePolicy
	XSet.setAutoDelete	XSet_SetAutoDelete
	XSet.applyShredPolicy	XSet_ApplyShredPolicy
	XSet.setShred	XSet_SetShred
	XSet.applyStoragePolicy	XSet_ApplyStoragePolicy
	XSet.getActualRetentionDuration	XSet_GetActualRetentionDuration
	XSet.getActualRetentionEnabled	XSet_GetActualRetentionEnabled
	XSet.getActualAutoDelete	XSet_GetActualAutoDelete
	XSet.getActualShred	XSet_GetActualShred
Import/Export	XSet.openExportXStream	XSet_OpenExportXStream
	XSet.openImportXStream	XSet_OpenImportXStream
Async I/O	XSet.submitJob	XSet_SubmitJob
	XSet.haltJob	XSet_HaltJob
	XSystem.asyncOpenXSet	XSystem_AsyncOpenXSet
	XSystem.asyncCopyXSet	XSystem_AsyncCopyXSet
	XSet.asyncOpenXStream	XAM_AsyncOpenXStream
	XStream.asyncRead	XStream_AsyncRead
	XStream.asyncWrite	XStream_AsyncWrite
	XStream.asyncClose	XStream_AsyncClose
	XSet.asyncCommit	XSet_AsyncCommit
Async Operations Management	XAsync.halt	XAsync_Halt
	XAsync.isComplete	XAsync_IsComplete
	XAsync.getXOPID	XAsync_GetXOPID
	XAsync.getStatus	XAsync_GetStatus
	XAsync.getXSet	XAsync_GetXSet
	XAsync.getXStream	XAsync_GetXStream

Table D.1 – C Method Name Mapping to XAM Architecture Specification

Type	Methods in Arch Spec	Methods in C API Spec
Async Operations Management (cont.)	XAsync.getXUID	XAsync_GetXUID
	XAsync.getBytesRead	XAsync_GetBytesRead
	XAsync.getBytesWritten	XAsync_GetBytesWritten
	XAsync.close	XAsync_Close
XUID	XUIDToString	base64_encode
	StringToXUID	base64_decode