What is XAM?

X-Access Method (or XAM) is a standard API between “Consumers” (application and management software) and “Providers” (storage systems) of reference information storage services.

XAM is an application layer that can be built on block and object devices, the latter is a natural fit.

XAM and OSD both standardized under SNIA

• Need to ensure they interoperate
Q4 2004 – IBM and EMC formulate a joint vision and begin work on a proposal

Q4 2005 – XAM Team donates v1.2 of XAM Spec to SNIA; Donation accepted, placed under control of FCAS TWG

mid Q3 2005 – XAM Team presents v1.1 to a select set of application ISVs, receives encouraging feedback

Q2 2006 – FCAS TWG counts >30 member companies, technical work in full progress

Q2 2006 – FCAS TWG counts >30 member companies, technical work in full progress

Today – FCAS TWG working on official API specification, targeted for Q4 2006

early Q3 2005 – v1.0 of XAM Spec available, HP, HDS, Sun endorse XAM, join XAM Team

Q4 2004 – IBM and EMC formulates a joint vision and begin work on a proposal
XAM Strategic Goals

Fixed Content has a long shelf life
  • e.g. regulations routinely require 7-20 years

Storage technology is transient
  • Content needs to be able to move across physical, geographical, technological, organizational boundaries

Content needs to be preserved in a self-describing way
  • Recognize “object” as a storage type, encapsulating data + metadata

XAM Enables:
  • applications to “migrate” across storage devices
  • “objects” to move between storage devices
  • “objects” to move between applications
An application uses the libxam.dll to ‘connect’ to a specified XSystem.
- A single application may connect to multiple XSystems simultaneously
- Multiple applications may connect to a single XSystem simultaneously

An XSystem is not identical to a vendor’s “storage box”, but a logical abstraction which should be viewed as ‘bag of storage’.

The application may be required to authenticate at the time the connection to an XSystem is established.

The application uses libxam.dll to store/retrieve “content objects” to/from the XSystem.

These “content objects” are bundles of data and metadata, and are called XSets.
XSet Fields

**Properties**
- “Simple” Types (Boolean, Uint64, Float64, String, DateTime, XUID)
- Type checked/enforced by Storage System
- Manipulated via “Property Get/Set” Methods

**Streams**
- Bytestreams, bound in Length
- Type assumed to be a valid MIME-type, but not checked/enforced by Storage System
- Manipulated via Posix-style I/O Methods (e.g. open, read, write, close)

-Each Field Has Four Basic Attributes:
- Type – stype for Properties, any other MIME-type for streams
- Length – The actual size of the field’s value
- Readonly – Flag indicating whether field is modifiable by applications
- Fixed – Flag indicating whether field is Fixed/Variable content
- Manipulated via “Attribute Get/Set” methods
XSet Field Names

Fields arranged within XSet in a hierarchical naming structure (Field Namespace)
- Examples:
  - `org.snia.xam.xset.retentionClass`
  - `com.emc.legato.email.fromHeader`
  - `org.aiim.ecm.standardField`

A Special Branch of the Field Namespace Reserved as ‘system section’
- Branch xam.* reserved for XAM Storage Vendors
- Applications may not create fields in this branch, but they may selectively read/edit/modify Fields
- A sub-branch (xam.org.snia.*) used for ‘standardized’ and ‘interoperable’ XAM Storage System Fields
- Examples:
  - `xam.org.snia.xam.xset.ctime`
  - `xam.com.emc.centera.hashid`
XAM Jobs

An XSystem is capable of running ‘Jobs’ within the storage system. Such ‘Jobs’ may include (but are not limited to):

- Index/Search/Query
- Bulk Delete

An application defines, initiates, controls and consumes the output of a running Job using a standard XSet.

- XSets have a dual nature – they server as both ‘content objects’ as well as ‘XAM Jobs’
- A Job may continue executing inside the XSystem even after the initiating application disconnects from it

In order to initiate a Job, an application:

- Creates an empty XSet
- Writes the Job’s input parameters/data into certain fields(*)
- Calls Xset.JobSubmit()
- Reads the Job’s output parameters/data from certain fields(*)

(*) the names and contents of these fields are job-specific
In order to run a Query, an application:

- Creates an empty XSet
- Specifies an SQL-like statement in a pre-defined field
- Calls Xset.JobSubmit()
- Reads the Query output from a pre-defined XStream

**Query Input Fields**

- Application specifies an XQL statement, e.g.
  - "select X where Y"

**Query Output Fields**

- XSystem creates an XStream within the same XSet
- This XStream contains the query output data

The application may read the output XStream while the query is still in progress

This XSet (with its input/output) may be persisted in the XSystem just like any other XSet
Mapping XAM to OSD
Mapping of Objects
Mapping of Functions
Example 1

Application:
"Find all records for patient X after 4/7/2006"

XAM Query:
SELECT WHERE org.nema.medical.patient_name="X" and org.nema.medical.record_date>"4/7/2006"

OSD:
OSDv2 QUERY command where attribute.name="X" and attribute.create_time>4/7/2006
Mapping of Functions

Example 2

Application:

“Destroy all records that have expired"

XAM:

SELECT WHERE org.nema.medical.record_date<“1/1/1999"

OSD:

–OSDv2 QUERY command where
attribute.create_time<1/1/1999

–OSDv2 REMOVE MEMBER command
Summary

– OSD is a good match for XAM
– Current OSD spec can be used as is if XAM properties are not to be interpreted
  • If they are, then new attributes need to be defined
– Details still need to be worked out