eXtensible Access Method (XAM)

- a new fixed content API

Mark A Carlson, SNIA Technical Council, Sun Microsystems, Inc.
Scott Ostapovicz, XAM SDK TWG, EMC
Topics

- What is Fixed Content?
- What is MetaData?
- What is SNIA doing?
- What is XAM?
- XAM API
- XAM SDK
What is Fixed Content?

- A type of data classification that indicates the bits are no longer changing
  - Classifying this way enables storage systems to meet the requirements of this type of data
- Most data is created “fixed”
  - Photos, videos, published/emailed documents, etc.
- 70-90% of data becomes fixed at some point
  - Even transactional data becomes fixed typically within a week
- Fixed content data is GROWING at 90% year over year
What is Metadata?

- Metadata allows for the creation of self-describing objects
- Self-describing objects enable content portability across client applications
- Metadata and location independence enable ILM across the managed storage resources
- Intelligent decisions can be made to allocate specific content types to specific storage resources
The need for MetaData Standards

- Which can contains corn?
- Open the cans.
- How much does it cost?
- Ask the clerk.
- How many calories does it have?
- Ask the vendor.
- How does the store automatically manage inventory?
- They can't.
Standardized labeling allows multiple vendors to consistently represent information to consumers.

Extended labeling for LOB uses.
Can access Email metadata and, without the help of the Email Service, analyze whether the sender is allowed to send to the recipient. For example, a stock analyst may not be allowed to send information to a trader.

Email object stored by XAM SDK

```
com.acme.email.from = "bugs bunny"
com.acme.email.from.role = "analyst"
com.acme.email.to = "daffy duck"
com.acme.email.to.role = "trader"
com.acme.email.subj = "what’s up doc?"
com.acme.email.numattach = 2
{ Email contents }
{ Attachment #1 }
{ Attachment #2 }
```

XAM specifies how metadata is represented, but not the actual metadata field names and values.

Further work is needed to standardize metadata names and allowed values for application domains like Email, Health, and Document Management.
SNIA XAM Standard Activities

- **The SNIA XAM Initiative** is chartered to drive adoption of XAM specification, and ensure that the specification fulfills market needs for a fixed content data management interface standard.

- **The SNIA Fixed Content Aware Storage Technical Work Group (FCAS TWG)** is the center of technical activities related to new application-level interfaces for storage of unchanging data (fixed content) and associated metadata.

- **The SNIA Software Development Kit Technical Working Group (XAM SDK TWG)** is chartered to develop SNIA Software which implements current and future versions of the XAM Specification(s).

Your partners and competitors are already participating.

Don’t be left out!
“Information independence for applications and storage”
XAM makes this possible

As seen at SNW Spring
Multi-Vendor demonstration based on XAM

Commercially Available Applications
- Records & Documents (Vignette)
- Disk Extender (EMC)
- RIM4DB/Outerbay (HP)
- Photo Editor (Sun)

Custom Application
- XSET Browser
- XAM Query Tool

Contributed Utilities

As seen at SNW Spring
Multi-Vendor demonstration based on XAM
The XAM initiative is

a SNIA Initiative

driven by the storage industry

to define and promote adoption of a standard application programming interface (the XAM API)

between “Consumers” (application and management software)

and “Providers” (storage systems)

of Fixed Content storage services
What is XAM?

- XAM is a SNIA Architecture
  - The XAM Architecture spec defines the normative semantics of the API for use by applications and implementation by storage systems
- XAM is an Application Programming Interface (API)
  - The XAM Java API spec defines the binding of the XAM Architecture to the Java Language
  - The XAM C API spec defines the binding of the XAM Architecture to the C Language
- XAM is SNIA Software
  - The XAM SDK provides a common library and reference implementation to promote widespread adoption of the standard
Why XAM?

The industry will benefit from a standardized access method to Fixed Content.

**Application Vendors want:**
- Annotate Data with associated Metadata
- Indicate basic Storage Management Policies
- Speak same language to all types of Devices
- Manipulate billions if not trillions or “records”

**Storage Vendors want:**
- Application Support for their Products
- Efficiently Store Application Data and Metadata
- Integrate Basic Storage Management Capabilities
- Manage billions if not trillions of “records”

**End Users want:**
- Choices between Application Vendors
- Choices between Storage Vendors
- Easy migration between vendors/technology
- Compliance, Scalability, Performance, $/GB, TCO
XAM is a Data Storage Interface

Resource Domains are a way of classifying services into specific areas that each deal with a different aspect of the problem.

An information domain application creates data and associates MetaData with it.

Certain Data Storage Interfaces can accommodate both Data and MetaData (XAM, Filesystems with extended attributes).

MetaData aware Data Services interpret Data System MetaData as the requirements for it's lifecycle and implement policies for retention, placement, lifecycle, etc.

Other Data Storage interfaces (based on blocks or objects) provide virtualized Containers for the Data bits and the management of those containers.

Storage services are employed to meet those requirements at this point in the data's lifecycle, however the storage services are unaware of the data's requirements.
XAM Data Storage Interface

- XAM is the first interface to standardize system metadata for retention of data
- XAM implements the basic capability to Read and Write Data (through Xstreams)
- XAM has the ability to locate any XSet with a query or by supplying the XUID
- XAM allows Metadata to be added to the data and keeps both in an XSet object
- XAM uses and produces system metadata for each XSet
- For example Access and Commit times (Storage System Metadata)
- But it also uniquely specifies Data System Metadata for Retention Data Services
- XAM User metadata is un-interpretable by the system, but stored with the other data and is available for use in queries
- Given this we can see that XAM is a data storage interface that is used by both Storage and Data Services (functions)
XAM System MetaData

- XAM specifies *property fields* that are interpreted by an Xsystem as System MetaData
- It does this by using a reserved field namespace
  - `.xam.*`, `.xsystem.*` and `.xset.*` are reserved
- Either the Content Application or a separate Information Management application can manipulate these fields to cause the underlying system to treat the data according to the specified requirements
  - i.e. Retention
Other standard data storage APIs have the ability to deal with metadata as well (POSIX filesystems)

- POSIX specifies standard system metadata as part of the data storage interface:
  - File times, Permission (including ACLs), owner, group, etc.
- This metadata is maintained and used (interpreted) by the storage services that implement the API
  - Thus we call it **storage system metadata**
- The functions that are controlled by this metadata govern the storing and retrieval of the data through the interface
- These functions are described in the abstract as **storage services**
Metadata

➢ Storage services may provide functions for metadata as part of the data storage interface.
  ➢ This is an important capability for managing Data Resources (as opposed to managing Storage Resources).

➢ The metadata may be managed by the storage service, managed by data services, or un-interpreted by either.
  ➢ System metadata that is managed by storage services are those properties of a data element that pertain to the primary functions of storing and retrieving the data.

➢ We call this *storage system metadata*, as it is used and managed by storage services.
  ➢ Other system and user metadata may be preserved on the basis of individual data elements, but is not interpreted by the storage services.
Metadata for Data Services

- Metadata available through the data storage interface may also be managed by data services.
- This data service metadata can be used by data services to provide differentiated value to individual data elements.
- The model or schema for data service metadata may be defined by each data service and may be standardized.
Metadata in Information Services

- The role of metadata in information services is as a communication mechanism with the underlying storage services and data services.

- Information services are primarily concerned with the data service system metadata as a means to convey the data’s requirements to the underlying data services.

- An information service may also interpret user metadata for purposes of data classification.

- An information service can create its own user metadata that is un-interpreted by the underlying services for its own use.
The Resource Domain Model

This model shows the logical layering of the different domains and the role of policies for each domain. The services in each domain play a different role, but leverage common, standard interfaces.
XAM SDK TWG Charter

- Develop SNIA Software that implements the XAM Library.
- Develop SNIA Software that implements a Reference Vendor Implementation Module (VIM) on top of an existing filesystem.
- Develop sample XAM Client Applications as SNIA Software to provide simple unit tests for portions of the XAM Specification(s).
- Develop documentation as appropriate for the above deliverables.
The XAM SDK is dynamically linked by each application wishing to connect to and use XSystems.

The XAM SDK includes several components:

- **XAM Library**, which implements the XAM API functions
- **Reference VIM**, which implements the reference behavior of an XSystem
- a framework which allows plug-able **Vendor VIMs**
- optional **XAM Toolkit** Libraries for convenience functions
Proliferation Questions

SDK Ecosystem

- Standardization Process
- Development & QA
- Integration & Distribution
- Interoperability Certification
- Licensing Schema
- Support & Maintenance
The Low Hanging Fruit

- XAM API Spec
- XAM Toolkit
- XAM Library
- Reference VIM
- Vendor A VIM
- Vendor X VIM
- XAM Arch Spec
- VIM API Spec

Developed, Supplied & Supported by any 3rd Party (incl. Vendors, ISVs etc)

Shipped with Operating Systems

Supplied & Supported by Individual Vendors of XAM Storage Systems
XAM SDK Proliferation - Summary

SNIA’s “FCAS TWG” maintains and periodically publishes set of normative XAM standard specs

SNIA’s “XAM Software TWG” develops and maintains beta-quality ‘Gold’ Distribution’ of XAM SDK under BSD License

SNIA’s Member Companies (e.g. EMC, IBM, HP, HDS, ...) derive their individual product-quality XAM SDK Derivatives from SNIA’s ‘Gold Distribution’

ISVs Integrate and certify their apps with a chosen Member Company’s XAM SDK Distribution

ISVs, Member Companies ship their products to End-Users with certified interoperability guarantees
The XAM SDK
developing XAM Library software

- XAM consists of a set of components.
  - The ‘topmost’ library will contain the public XAM interfaces; thus, only the topmost library will be directly referenced by applications that wish to integrate with the XAM API.
  - 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 reference 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.

- Components will be produced in both C/C++ and Java
Design Goals

- Provide a generic interface for applications
  - XAM API methods have the same syntax and semantics without regard to the underlying storage. No methods exist 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
  - Keep the interface simple and small (e.g., have as few API methods as possible, and keep these methods easy to use and understand), while ensuring 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 scalability), then a new method is not created for that function.

- Expose no implementation detail
  - Do not expose any internal functionality that would serve to place restrictions on storage system vendors.
Organization of the XAM Architecture

- XAM Interface semantics are organized around objects
  - Primary objects
    - XAM Library
    - XSystem
    - XSet
  - Secondary objects
    - XStream
    - XIterator
XAM Primary Objects

- Load library
- Unload library
- Connect to an XSystem
- Close an XSystem
- Open/Create an XSet
- Close an XSet
- Field Editing

XAM Library

XSystem
- XSystem authentication
- XSet administration

XSet
- Job control
- XSet import/export
- XSet commit

Field Editing
**XAM Secondary Objects**  
*(with relationship to Primary Objects)*

- XSet, XSystem, or XAM Library
- Field Editing
- Open/Create an XStream
- XStream Ops
- XStream
- Open XIterator
- Close
- XIterator
- XIterator ops
Organization of the XAM Architecture

- XAM Interface semantics are organized around objects
  - Primary objects
    - XAM Library
    - XSystem
    - XSet
  - Secondary objects
    - XStream
    - XIterator
Elements of Primary Objects

- **XAM Library object**
  - No constructor is available
  - This is a singleton
  - This must be available as a static object or thru the use of a static accessor method
  - Contains fields

- **XSystem object**
  - No constructor is available
  - This must be available by calling a factory method on the XAM Library object
  - Contains fields

- **XSet object**
  - No constructor is available
  - This must be available by calling a factory method on the XSystem object
  - Contains fields
Elements of Primary Objects

- **XAM Library object**
  - No constructor is available
  - This is a singleton
  - This must be available as a static object or thru the use of a static accessor method
  - **Contains fields**

- **XSystem object**
  - No constructor is available
  - This must be available by calling a factory method on the XAM Library object
  - **Contains fields**

- **XSet object**
  - No constructor is available
  - This must be available by calling a factory method on the XSystem object
  - **Contains fields**
UML of the Primary Object
Elements of Secondary Objects

XIterator object
- No constructor is available
  - This must be available by calling a factory method on a Primary object (implements Field Container)

XStream object
- No constructor is available
  - This must be available by calling a factory method on a Primary object (implements Field Container)
UML of the Secondary Objects

```
<interface>
org.snia.xam.XStream
+tell()
+seek()
+write()
+read()
+close()
+abandon()
</interface>

<interface>
org.snia.xam.Logger
+setLevel()
+getLevel()
+fatal()
+error()
+warn()
+info()
+trace()
+debug()
</interface>
```
The VIM Interface roughly maps to the public XAM API

- Each object in the XAM API should have an analog in the VIM Interface
- Each method in the XAM API should have an analog in the VIM Interface

NOTE: For Java - the VIM instances are interacted with “directly” by the application (no need to proxy/copy)
Control Flow

- Note that in all cases, the VIM is accessed thru the XSystem
  - There is no public interface that exposes the VIM to the application.
- Possible control flows:
  - Objects created by a VIM are directly passed to the application.
  - Objects created by a VIM are decorated by the XAM Library and the references are indirectly passed to the application; the XAM Library thus holds references to the objects and resolves references for application.
Direct control (used by Java)

1. Application
   - `connect()`

2. XAM Library
   - XAM Library locates VIM from XRI, creates instance of VIM
   - `new VIM instance`
   - `new XSystem`
   - `authenticate()`
   - `authenticate return`
   - `createXSet()`

3. Java VIM
   - `new XSystem`
   - `XSystem`

4. XSystem
   - `XSystem`
   - `XSystem`
   - `new XSet`
   - `XSet`

5. XSet
   - `createProperty(int)`
   - `createProperty return()`
   - `XSet`
Indirect Control (used by C/C++)
As noted before, the application binds to the XAM API

Applications should never bind to the VIM interface!

It is the responsibility of the XAM Library to call into the VIM, not the application.

The VIM interacts with the Storage System

The XAM Library never interacts directly with the underlying Storage System; all ‘communication’ is routed thru the VIM
XAM Application software stack (C/C++)

- C/C++ Application
  - XAM Java API (JNI)
    - XAM C API
      - XAM Library logic (C++)
        - ‘outgoing’ VIM C API
          - VIM Java API (JNI)
            - VIM C API
              - VIM C
XAM Application software stack (Unified)

- C/C++ Application
  - XAM C API
  - XAM Library logic (C++)
  - ‘outgoing’ VIM C API
  - VIM C API
  - VIM
- Java Application
  - XAM Java API
  - XAM Library logic (Java)
  - ‘outgoing’ VIM Java API
  - VIM Java API
  - VIM Java
A ‘stackable’ VIM

- XAM places limited constraints on the implementation of the VIM; only that it must implement the VIM interface
  - A VIM is allowed to call other VIMs.
- A ‘stackable’ VIM is a VIM that is capable of calling into the VIM Interface of other VIMs.
- This fully unifies the software model
  - Any VIM can be used with any XAM Library if an appropriate stackable VIM exists.
XAM Application software stack
(Fully Unified)
HTTP VIM Example

- The HTTP VIM is an example of a stackable VIM.
Where To Go

- SNIA XAM Home
  - http://www.snia.org/xam

- SNIA FCAS TWG
  (XAM Technical Work Group)

- SNIA XAM SDK TWG
  (XAM SDK Technical Work Group)