INTERNATIONAL STANDARD

ISO/IEC 23681

First edition
2019-05

Information technology — Self-contained Information Retention Format (SIRF) Specification

Reference number
ISO/IEC 23681:2019(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by SNIA (as SIRF Specification V1.0) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Many organizations now have a requirement to preserve and maintain access to large volumes of digital content indefinitely into the future. Regulatory compliance and legal issues require preservation of email archives, medical records and information about intellectual property. Web services and applications compete to provide storage, organization and sharing of consumers' photos, movies, and other creations. And many other fixed-content repositories are charged with collecting and providing access to scientific data, intelligence, libraries, movies and music. A key challenge to this need is the creation of vendor-neutral storage containers that can be interpreted over time.

Archivists and records managers of physical items such as documents, records, etc., avoid processing each item individually. Instead, they gather together a group of items that are related in some manner — by usage, by association with a specific event, by timing, and so on — and then perform all of the processing on the group as a unit. The group itself may be known as a series, a collection, or in some cases as a record or a record group. Once assembled, an archivist will place the series in a physical container (e.g., a file folder or a filing box of standard dimensions), mark the container with a name and a reference number and place the container in a known location. Information about the series will be included in a label that is physically attached to the container, as well as in a "finding aid" such as an online catalog that conforms to a defined schema and gives the name and location of the series, its size, and an overview of its contents.

This document proposes an approach to digital content preservation that leverages the processes of the archival profession thus helping archivists remain comfortable with the digital domain. One of the major needs to make this strategy possible is a digital equivalent to the physical container — the archival box or file folder — that defines a series, and which can be labelled with standard information in a defined format to allow retrieval when needed. Self-contained Information Retention Format (SIRF) is intended to be that equivalent — a storage container format for a set of (digital) preservation objects that also provides a catalog with metadata related to the entire contents of the container as well as to the individual objects and their interrelationship. This logical container makes it easier and more efficient to provide many of the processes that will be needed to address threats to the digital content. Easier and more efficient preservation processes in turn lead to more scalable and less costly preservation of digital content.

SIRF components, use cases and functional requirements were defined in [1] SIRF use cases and functional requirements, working draft — version 0.5a and further described in [2] "Towards SIRF: Self-contained Information Retention Format." This document goes one step further and details the actual metadata, categories and elements in the container's catalog. The document also describes how the SIRF logical format is serialized for storage containers in the cloud and for tape based containers. The SIRF serialization for the cloud is being experimented with OpenStack Swift object storage, and the implementation is offered as open source in the OpenSIRF initiative[3].

Creating and maintaining the SIRF catalog requires executing data-intensive computations on the various preservation objects including fixity checks, data transformations. This can be done efficiently via executing computational modules — storlets — close to where the data is stored. The benefits of using storlets include reduced bandwidth (reduce the number of bytes transferred over the WAN), enhanced security (reduce exposure of sensitive data), costs savings (saving infrastructure at the client side) and compliance support (improve provenance tracking). The Storlet Engine[4] (see "Storlet Engine for Executing Biomedical Processes within the Storage System") is an engine to support such storlets computations in secure sandboxes within the storage system, and can be used to create and maintain SIRF containers.

Information technology — Self-contained Information Retention Format (SIRF) Specification

1 Scope

This document specifies the Self-contained Information Retention Format (SIRF) Level 1 and its serialization for LTFS, CDMI and OpenStack Swift.

This document proposes an approach to digital content preservation that leverages the processes of the archival profession thus helping archivists remain comfortable with the digital domain.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

ISO/IEC 17826:2012, *Information technology — Cloud Data Management Interface (CDMI)*

ISO/IEC 20919:2016, *Information technology — Linear Tape File System (LTFS) Format Specification*

Self-contained Information Retention Format (SIRF) use cases and functional requirements, working draft — version 0.5a, SNIA, September 2010, http://www.snia.org/tech_activities/publicreview/SIRF_Use_Cases_V05a_DRAFT.pdf

JSON. ECMA-404, The JSON Data Interchange Standard. http://json.org

OpenSIRF. http://github.com/opensirf

OpenStack Swift. http://swift.openstack.org

PREMIS. PREservation Metadata: Implementation, Strategies, http://www.loc.gov/standards/premis

Rabinovici-Cohen S., Henis E., Marberg J., Nagin K. Storlet Engine for Executing Biomedical Processes within the Storage System", Proceedings of the 7th International Workshop on Process-oriented Information Systems in Healthcare (ProHealth), September 2014, Eindhoven, the Netherlands

Rabinovici-Cohen S., Baker M.G., Cummings R., Fineberg S., Marberg J. Towards SIRF: Self-contained Information Retention Format", Proceedings of the Annual International Systems and Storage Conference (SYSTOR), May 30-June 1, 2011, Haifa, Israel. https://www.research.ibm.com/haifa/projects/storage/datastores/papers/systor56-rabinovici-cohen.pdf

Rabinovici-Cohen S., Cummings R., Fineberg S. Self-contained Information Retention Format for Future Semantic Interoperability", Proceedings of the 4th International Workshop on Semantic Digital Archives (SDA), September 2014, London, UK

W3C Prov Model Primer http://www.w3.org/TR/prov-primer/

3 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

4 Business Case

While no one wants to lose their digital content, the cost of maintaining integrity and access is significant, in both money and effort. And unlike paper based content, the lifespan of digital content can be very short unless if proactive steps are being taken to protect it. The use of a storage container format like SIRF adds little expense and greatly increases the sustainability of data. However, this is not adequate unless if the cost of preserving content is less than the (potential) cost of losing it.

In a business context, there are three major reasons why content is preserved. These are: to preserve history, to mitigate risk or meet a legal mandate, and for future value of information. One or more of these may apply, and the amount an entity is willing to spend will differ depending on how well these reasons are aligned with the business goals of an organization.

One of the main reasons why people and organizations preserve content is to preserve history. In the case of an individual, it may be photos, videos, and other content preserving one's life history. In a business context, libraries, national archives, historians, and others have a primary mission to preserve history.

This should mean that information under the control of these organizations or individuals would be well protected. However, reality dictates that:

1) Organizations (and individuals) have limited resources, and they have to make choices how much to invest in preservation. Typically, this happens with the organization's knowledge of what trade-offs are being made. However, in the case of digital content, the choice not to invest in preservation will typically result in loss.

2) Organizations cannot preserve everything as the cost would be prohibitive. However, it is difficult to predict what will be of historic value. Sometimes important content is lost simply because its value was not known at the time.

3) Lack of skilled and experienced personal may also result in loss, especially in situations where simple solutions do not exist.

Another often cited reason for preserving data is for "risk mitigation", or in some cases for "legal mandate". These are closely related reasons because legal mandate is often looked at through the lens of legal risk. In other words, a mandate that is not enforced or whose penalty is small is less of a risk than a mandate with a larger penalty.

For example, consider government entities like a national or state archive. Legislators require those entities to keep records for a prescribed period of time. The penalty for losing those records may include loss of job, loss of funding, or even criminal penalties. Since preservation is directly funded, there is little excuse for losing information. However, even in those cases, underfunding, and lack of expertise may result in loss. Further, when those agencies preserve historical information, the mandate and funding will not allow them to keep everything. Therefore, even these archivists are required to make choices about what to keep.

Another often cited legal mandate is in healthcare, where medical organizations are required to retain information for the lifetime of a patient. This seems like a difficult requirement, especially since records are often maintained in private doctors' offices and other places that may not exist 50 years or 75 years into the future. Anecdotal evidence shows that medical records are not maintained that long. So, why is this happening? It is because records retention is expensive, and there are no penalties for losing information. That is not to say that doctors and hospitals don't try, rather they won't spend the necessary money to ensure that records are not lost.

The private sector is similar to healthcare. Preservation is seen as a cost, so it only makes sense where the return on investment is positive. For example, businesses are very good at keeping recent tax records, because they know that if they don't, the tax authority may levy fines. In the US, businesses retain emails, because they know judges will fine them if they don't. They even implement searchable

archive systems, because they know that it will save money when they are forced to produce documents responsive to a lawsuit. Note that this is different outside of the US, where judges often do not levy fines or force companies to spend large sums of money on legal discovery.

When private sector companies do preserve information, they are typically focused on risk mitigation, not preservation. If a company can legally delete information, they often will, because it eliminates the chance that it can be used against them. If information is purely a risk, and a company is not in the business of preservation, why would they keep it? The obvious answer is that they will keep information that is valuable, and other information will not be kept. Value can result from future revenue, cost savings, technical advantage, etc.

Regarding future value of information, one obvious example is in the entertainment industry. Movies, TV shows, music, and other content can be re-sold and repurposed decades after its creation. This can result in many dollars in revenue. So not surprisingly, organizations like the Motion Picture Experts Group are at the leading edge of digital preservation. Entertainment companies spend significant amounts of money retaining their content so that they will have it available to repurpose. However, this does not mean they can retain everything. With the advent of digital movie production, the amount of data that can be generated during the creation of a single film is immense. Therefore, even here where future value is tangible, some hard choices need to be made.

In other industries, the mandate may not be as clear. Are design documents from an existing product valuable? What about a retired product? What about research leading up to a product design? These things may be needed for risk mitigation, but what about for use in future products? Companies make decisions about these kinds of Intellectual Property content every day, and more times than not, the data is either actively destroyed or lost due to inaction. The reason for this is actually because its value (beyond risk mitigation) is unknown, so it's not clear how much a company should invest in retaining the information.

So, how does SIRF help? SIRF brings down the expense of preservation, because data can remain accessible even if the software that created the data no longer exists. This is because the stored data is designed to be understandable, and does not need specialized software to interpret it. SIRF reduces the complexity of logical and physical migration, making it easier for businesses to justify. By using SIRF today, it becomes possible to retain more information, and to retain information with a lower perceived future value. This is unlike proprietary and undocumented formats, which become useless soon after a business stops paying for support.

5 Specification Overview

5.1 Container Components

Figure 1 illustrates the SIRF container, which includes the following components:

— A magic object that identifies whether this is a SIRF container and gives its version. The magic object is independent of the media and has an agreed defined name and a fixed size. It also includes the means to access the SIRF catalog (for example, the catalog's location).

— Preservation objects that contain the actual data to be preserved. An example preservation object can be the OAIS Archival Information Package (AIP). The container may include multiple versions of a preservation object and multiple copies of each version, but each specific preservation object is generally immutable.

— A catalog that is updateable and contains metadata needed to make the container and its preservation objects portable and accessible into the future without relying on metadata external to the storage subsystem.

While the semantics of traditional storage systems include only limited standardized metadata about each object, SIRF provides for the rich metadata needed for preservation and ensures its grouping with the data. This rich metadata is defined in the catalog in a logical format to allow its serialization for different storage technologies.

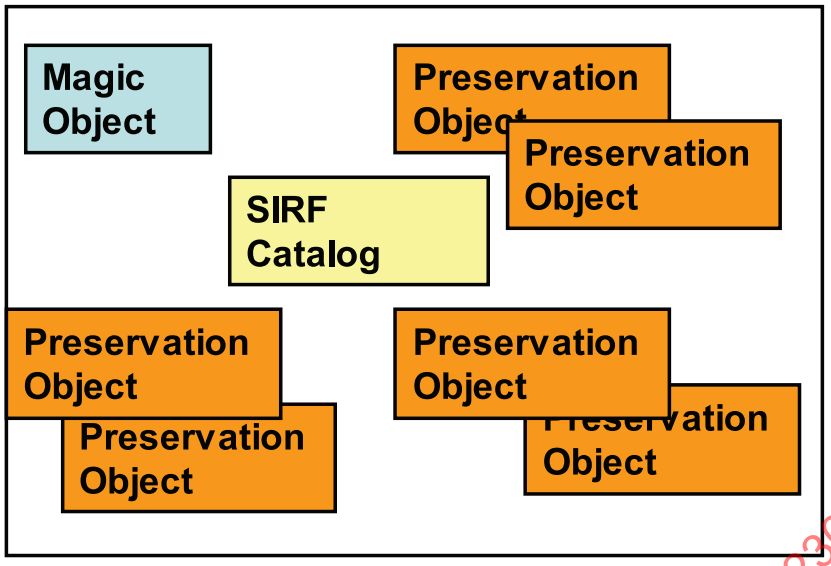

Figure 1 — SIRF Components

5.2 SIRF Catalog

The SIRF catalog is an object that includes metadata about the Preservation Objects (POs) in the container and their relationship. It has a well-defined standardized format so it can be understandable in the future. The SIRF catalog is separated from the metadata contained in the POs themselves because a strict standardized format is difficult to impose on the POs that are generated by different applications and domains. Additionally, The SIRF catalog level 1 includes metadata that is not included in the PO, e.g., fixity value of the whole PO. Including this metadata within the PO changes the fixity value of the PO making this metadata inherently incorrect.

The SIRF catalog includes metadata related to the whole container as well as metadata related to each preservation object within the container. Both types of metadata are organized into categories.

The metadata for the whole container includes the following categories:

1 Container Information

- 1.1 Specification
- 1.2 Container ID
- 1.3 State
- 1.4 Provenance
- 1.5 Audit Log

The metadata for all the preservation objects set is aggregated under:

2 Objects Set

The metadata for each preservation object includes the following categories:

3 Object Information

- 3.1 Object IDs
- 3.2 Related Objects
- 3.3 Dates

3.4 Packaging Format

3.5 Fixity

3.6 Retention

3.7 Audit Log

3.8 Extension

Examples of SIRF Catalogs are shown in Annex A: (informative) XML schema for the SIRF catalog, Annex B: (informative) Sample XML catalog, and Annex C: (informative) Sample JSON catalog.

5.3 Metadata Units

This document describes the specific metadata units in the various categories. Each category includes several elements in which each element may be composed of several attributes. The document also provides a hierarchical representation of the metadata units. The notation of the hierarchical representation is based on PREMIS[5] (PREservation Metadata: Implementation, Strategies) when possible and includes the following notions:

Repeatability (R): A metadata unit designated as "Repeatable" (R) can take multiple values. It does not mean that a repository must record multiple instances of the metadata unit. Similarly, a metadata unit can be designated as Not Repeatable (NR).

Optionality (O): Whether a value for the metadata unit is optional (O) or mandatory (M). Values for mandatory metadata units are required while values for optional metadata units are encouraged but not required.

SIRF serialization for CDMI/LTFS/Swift specify how a CDMI container, LTFS Tape or Swift container can become also SIRF-compliant. A SIRF-compliant CDMI container, LTFS Tape or Swift container enables future CDMI/LTFS/Swift clients "understand" containers created by today's CDMI/LTFS/Swift clients although the properties of the future client is unknown to us today. By "understand", it means that clients can identify the preservation objects in the container, the packaging format of each object, its fixity values, etc. (as defined in the SIRF catalog). This document also includes sections on SIRF serialization for CDMI (Section 6), SIRF serialization for LTFS (Section 7) and SIRF serialization for OpenStack Swift (Section 8).

SIRF includes metadata about the storage container, to help "understand" the container information in the future. No single technology will be usable over the time spans mandated by current digital preservation needs. SNIA CDMI, Swift and LTFS technologies are among best current choices, but are good for perhaps 10 years to 20 years. SIRF provides a vehicle for collecting all of the information that will be needed to transition to new technologies in the future, and it can be serialized for the future technologies as they come.

6 Container Information Metadata

6.1 Specification Category

This category includes information about the SIRF specification used for this container. As the **specification may evolve over time and distinct containers may use different SIRF specifications, it is** denoted in the SIRF catalog the specification used, and the SIRF level. SIRF level 2 specification defines more detailed metadata in the container.

The elements of the Specification category are:

— **Specification ID** (containerSpecificationIdentifier) — the specification identifier e.g., "SIRF-1.0"

— **Specification Version** (containerSpecificationVersion) — the specification version e.g., "1.0"

— **SIRF Level** (containerSpecificationSirfLevel) — the SIRF level that should be "1"

Hierarchical Representation

1 containerInformation (1-1: M, NR)

1.1 containerSpecification (1-1: M, NR)

1.1.1 containerSpecificationIdentifier (1-1: M, NR)

1.1.2 containerSpecificationVersion (1-1: M, NR)

1.1.3 containerSpecificationSirfLevel (1-1: M, NR)

6.2 Container ID Category

This category includes the container unique identifier and it has just one element:

— **Container ID** (containerIdentifier) — the container unique identifier, e.g., the tape id or cloud container id

The Container ID element is composed of the following attributes:

— **Container Identifier Type (containerIdentifierType)** — a designation of the naming authority and the domain within which the object identifier is unique.

— **Container Identifier Locale (containerIdentifierLocale)** — the locale of the identifier based on the Internet Assigned Numbers Authority (IANA).

— **Container Identifier Value (containerIdentifierValue)** — a Unicode/UTF-8 string for identifier actual value.

Hierarchical Representation

1 containerInformation (1-1: M, NR)

1.2 containerIdentifier (1-1: M, NR)

1.2.1 containerIdentifierType (1-1: M, NR)

1.2.2 containerIdentifierLocale (1-1: M, NR)

1.2.3 containerIdentifierValue (1-1: M, NR)

6.3 State Category

The state metadata is an indication of the progress of any activities that are to be carried out against a container. For example, if a container holds many preservation objects, state may indicate whether all of the objects intended for a container have been included or not. Or, state may indicate an in-process migration of a container.

The elements of the State category, as shown in Table 1, are:

— **State Type** (containerStateType)

— **State Value** (containerStateValue)

Table 1 — State category types and values

Type	Accepted Values	Use
INITIALIZING	TRUE	TRUE when the container is being initialized, i.e., the magic object, container provenance and/or catalog are being created
READY	ACTIVE FINALIZED	ACTIVE when the container is ready to receive new POs or catalog changes FINALIZED when the container is closed, read-only, and cannot receive more POs
NOT READY	DESTROYED ERROR	DESTROYED when the container has been destroyed and can no longer be read or modified. ERROR when there's a failure that cannot be specified using any other state type
MIGRATING	TRUE	TRUE when the data stored in a container are being moved to/from another container

The container state transitions occur depending on the current state and the action that is being taken in the container. Figure 2 shows the possible state transitions.

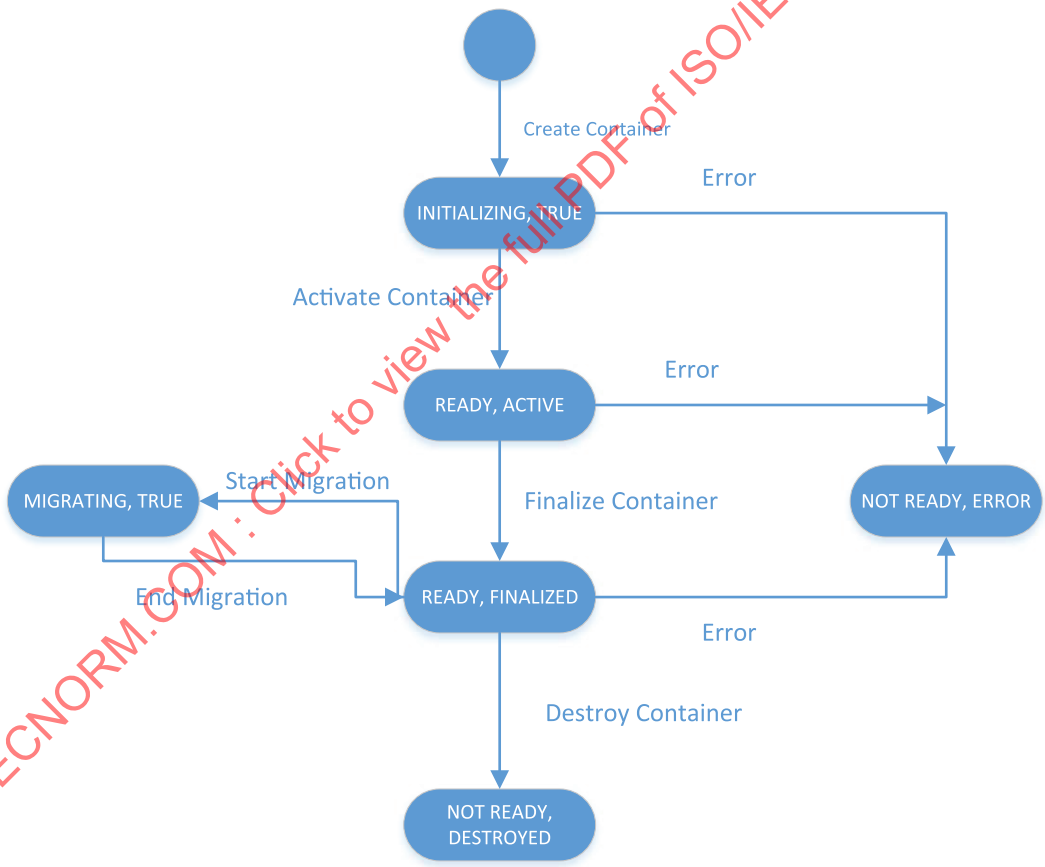

Figure 2 — Possible container states and transitions

Hierarchical Representation

1 containerInformation (1-1: M, NR)

1.3 containerState (1-1: M, NR)

1.3.1 containerStateType (1-1: M, NR)

1.3.2 containerStateValue (1-1: M, NR)

6.4 Provenance Category

Provenance is metadata describing the history of the information in a SIRF container (e.g., its origins, chain of custody, preservation actions and effects). The W3C provenance primer[6] (W3C Prov Model Primer) describes three perspectives for provenance information:

1) One perspective might focus on agent-centered provenance, that is, what people or organizations were involved in generating or manipulating the information in question. For example, in this perspective the provenance of the container can include the user that created the SIRF container.

2) A second perspective might focus on object-centered provenance, by tracing the origins of portions of the container. An example of this perspective is having the container assembled from content from other containers.

3) A third perspective one might take is on process-centered provenance, capturing the actions and steps taken to generate the information in question. For example, the container may have been generated by invoking a service to retrieve data from a database, then generating preservation objects, and finally storing the preservation objects with a specific application.

Regardless of the perspective from which provenance metadata is derived, it is critical for understanding the container, its history, its context and meaning. SIRF enables the preservation of container provenance information as part of a container's metadata.

The provenance information may vary depending on the type of information being preserved or its intended audience. In addition, it may be larger than what can reasonably be included in the catalog. Therefore, it is included in the catalog by reference, and the actual information is stored in another preservation object:

This category includes just one element:

— **Provenance Reference** (containerProvenanceReference) — reference to the object that contains the provenance

The container provenance information stored in SIRF may be in the W3C-PROV format, or any other well-known provenance format.

The containerProvenanceReference element is composed of the following attributes:

— **Reference Type (referenceType) —** whether internal referencing within the container or external referencing

— **Reference Role (referenceRole)** — the value should be "Provenance"

— **Reference Value (referenceValue) —** the unique identifier of the referenced provenance object. If the referenced object is an internal preservation object, the value will be its objectVersionIdentifier. objectIdentifierValue

Hierarchical Representation

1 containerInformation (1-1: M, NR)

1.4 containerProvenance (1-*: M, NR)

1.4.1 containerProvenanceReference (1-1: M, NR)

1.4.1.1 referenceType (1-1: M, NR)

1.4.1.2 referenceRole (1-1: M, NR)

1.4.1.3 referenceValue (1-1: M, NR)

6.5 Audit Log Category

The audit log is provided as a place for preserving any important information about how a container has been accessed or modified. Note that this category is for the audit log of the container, in contrast to the audit log described in the Object Information Metadata section. The extent and contents of an audit log depend on the needs of the specific preservation data store and its use case. Distinct domains have different audit logs regulations e.g., SEC is for the market domain, FDA is for the pharmatutical domain. In SIRF, audit logs are stored as SIRF preservation objects. Therefore, a container audit log information is stored as an object ID in the SIRF catalog.

This category includes just one element:

— **Audit Log Reference** (containerAuditLogReference) — reference to the object that contains the audit log

The containerAuditLogReference element is composed of the following attributes:

— **Reference Type (referenceType) —** whether internal referencing within the container or external referencing

— Reference Role (referenceRole) — the value should be "AuditLog"

— **Reference Value (referenceValue) —** the unique identifier of the referenced audit log object. If the referenced object is an internal preservation object, the value will be its objectVersionIdentifier. objectIdentifierValue

Hierarchical Representation

1 containerInformation (1-1: M, NR)

1.5 containerAuditLog (0-*, O, R)

1.5.1 containerAuditLogReference (1-1: M, NR)

1.5.1.1 referenceType (1-1: M, NR)

1.5.1.2 referenceRole (1-1: M, NR)

1.5.1.3 referenceValue (1-1: M, NR)

7 Object Information Metadata

The SIRF container includes multiple preservation objects (POs). The objectsSet is the element in the SIRF catalog that aggregates all the object information metadata of the various POs. Underneath the objectsSet are the various objectInformation — one for each PO. Sections 0 through 7.8 describe the categories, elements and attributes within the objectInformation.

Hierarchical Representation

2 objectsSet (1-*: M, R)

7.1 Object IDs Category

Identifiers (IDs) are used to identify a PO and to link to other POs. Managing identifiers over the long term raises issues such as:

— how to ensure uniqueness of identifiers over long term

— how to handle evolution of identifiers over time

— how to ensure scalability of identifiers

SIRF helps addressing these issues by enabling redundancy in identifiers and registering the evolution (genealogy) of POs. Hence, a PO in a SIRF container can have multiple identifiers as redundancy in identifiers increases the chances that at least one identifier will survive for the long term. Nevertheless, at any time, at least one of the identifiers should be persistent and unique.

When an identifier is qualified as unique, it should be unique across all types of storage including offline storage and online storage. There are various methods to generate unique identifiers, which may vary in the scope of uniqueness. Unique identifiers are sometimes based on external infrastructure. For example, the British Library unique identifiers are based on a secure timestamp provided by the British government. In another example, the unique identifier can be based on a domain name and include a vendor identifier.

Universal Unique IDs (UUIDs) are sometimes selected for creating unique identifiers. Note that in a long-term environment, there are additional challenges with managing UUIDs and ensuring their uniqueness. For example, there is a need to ensure that the same process and method for generating UUIDs is continually used, so that it can be guaranteed over long periods of time that no duplicate UUID would be generated. Also note that as of today, there are no means to enforce the methods of which the UUIDs are generated and to validate that they do not collide.

The elements of the Object IDs category are:

— **PO Name (objectName)** — non-unique identifier e.g., file name

— **PO Version ID (objectVersionIdentifier)** — unique identifier that identifies the specific version of the PO

— **PO Logical ID (objectLogicalIdentifier)** — a unique identifier that identifies the various versions that originate from the same ancestor

— **PO Parent ID (objectParentIdentifier)** — a unique identifier that identifies the parent PO from which this PO version was created. The Parent PO shares the same logical ID as the current PO, but has a different version ID. Parent ID can be added at a later stage, and it is not mandatory to specify it when the PO is created. This is because the parent PO may be stored in the container at a later stage or can be stored in another container.

A PO may have several copies that are bitwise identical. Different copies may reside in different containers. Each one of these copies may have a different copy ID, but their version ID, logical ID, and parent ID should be the same. Elements such as "copy ID" or "number of copies" are not maintained in the SIRF catalog, as maintaining them in each container and updating the catalog each time a copy is lost may be difficult and subject to inconsistency. It is hard to manage such an element, especially if some of the copies reside in other storage containers. For example, consider the case in which a copy of a PO is destroyed, and an update of the catalog of all containers was needed. Also, when a new copy is created instead, some implementations may choose to assign a new copy ID to the newly created PO, while other implementations may choose to assign the same copy ID as the destroyed one.

Each element in this category is composed of the following attributes:

— **Object Identifier Type (objectIdentifierType)** — a designation of the naming authority and the domain within which the object identifier is unique. Identifier values cannot be assumed to be unique across domains; the combination of *objectIdentifierType* and *objectIdentifierValue* should ensure uniqueness. Examples of naming authority and domain include: URL, DataCite DOI, ARK, UUID, vendor domain, DLC, DRS, Handle System HDL.

— **Object Identifier Locale (objectIdentifierLocale)** — the locale of the identifier based on the Internet Assigned Numbers Authority (IANA).

— **Object Identifier Value (objectIdentifierValue)** — a Unicode/UTF-8 string for identifier actual value. There is no limit on the objectIdentifierValue length in the architecture, but the implementation can have size limits. The objectIdentifierValue can sometimes give a hint regarding the classification of the data, but this is left to the implementation. Examples of identifiers values include: http://x.y.z, DOI:123456, urn::xx::dd, hdl:4263537.

Hierarchical Representation

The objectName includes all the non-unique identifiers of the object and there may be 0-* such identifiers namely this element is optional (O) and there may be multiple such elements repeating (R).

The objectVersionIdentifier includes the unique identifiers of this version of the object and may have 1-* such identifiers, with at least one such element mandatory (M). Multiple elements may repeat (R).

The objectLogicalIdentifier includes the unique identifier of all the versions originating from the same ancestor. This element is mandatory (M). There can be only one such element, which is not repeating (NR).

The objectParentIdentifier includes the unique identifier of the parent object. This element is optional (O), as not all objects have parents, but if it exists then there can be only one such element, which is not repeating (NR).

The hierarchical representation of the Object IDs category is as follows:

3 objectInformation (1-*: M, R)

3.1 objectIdentifiers (1-*: M, R)

3.1.1 objectName (0-*: O, R)

3.1.1.1 objectIdentifierType (1-1: M, NR)

3.1.1.2 objectIdentifierLocale (1-1: M, NR)

3.1.1.3 objectIdentifierValue (1-1: M, NR)

3.1.2 objectVersionIdentifier (1-*: M, R)

3.1.2.1 objectIdentifierType (1-1: M, NR)

3.1.2.2 objectIdentifierLocale (1-1: M, NR)

3.1.2.3 objectIdentifierValue (1-1: M, NR)

3.1.3 objectLogicalIdentifier (1-1: M, NR)

3.1.3.1 objectIdentifierType (1-1: M, NR)

3.1.3.2 objectIdentifierLocale (1-1: M, NR)

3.1.3.3 objectIdentifierValue (1-1: M, NR)

3.1.4 objectParentIdentifier (0-1: O, NR)

3.1.4.1 objectIdentifierType (1-1: M, NR)

3.1.4.2 objectIdentifierValue (1-1: M, NR)

3.1.4.3 objectIdentifierLocale (1-1: M, NR)

7.2 Dates Category

The dates category is used to keep a record of the last time a preservation object was modified and accessed (both optional), as well as when it was created (mandatory). Note that the modified time is optional since it is meaningless in a system that does not allow changes after creation. Accessed time is also optional since some systems do not actively track access, and because it may be costly in a system that is frequently accessed.

All date/time values are in the ISO 8601:2004 extended representation (YYYY-MM-DDThh:mm:ss .ssssssZ). The full precision shall be specified, the sub-second separator shall be a ".", the Z UTC zone indicator shall be included, and all timestamps shall be in UTC time zone. The YYYY-MM-DDT24:00:

00.000000Z hour shall not be used, and instead, it shall be represented as YYYY-MM-DDT00:00:00 .000000Z.

Hierarchical Representation

3 objectInformation (1-*: M, R)

3.2 objectDates (1-1: M, NR)

3.2.1 objectCreationDate (1-1: M, NR)

3.2.2 objectLastModifiedDate (0-1: O, NR)

3.2.3 objectLastAccessedDate (0-1: O, NR)

7.3 Related Objects Category

The Related Objects category is used to reference other POs in the SIRF container that relate to the current PO described. Examples of such related objects can be an object that associates context to the current object, or an object that is representation information (as defined in OAIS) of the current object.

The Related Objects category has one element:

— **Related Object Reference** (objectRelatedObjectsReference) — reference to the object that is related to current object

The objectRelatedObjectsReference element is composed of the following attributes:

— **Reference Type (referenceType) —** whether internal referencing within the container or external referencing

— **Reference Role (referenceRole) —** the type of relation to the current object, e.g., whether "context", "representation information", etc.

— **Reference Value (referenceValue) —** the unique identifier of the related object. If the object is an internal PO, the value will be its objectVersionIdentifier.objectIdentifierValue

Hierarchical Representation

3 objectInformation (1-*: M, R)

3.3 objectRelatedObjects (0-*, O, R)

3.3.1 objectRelatedObjectsReference (1-1: M, NR)

3.3.1.1 referenceType (1-1: M, NR)

3.3.1.2 referenceRole (1-1: M, NR)

3.3.1.3 referenceValue (1-1: M, NR)

7.4 Packaging Format Category

The Packaging Format category is used to denote the format of the manifest of the PO, e.g., PREMIS, XIP, XFDU. Different preservation objects in the SIRF container may have different packaging formats.

The Packaging Format category has one element:

— **Packaging Format Name** (objectPackagingFormatName) — a string that represents the format of the PO including format version. The string may also include the compression algorithm name if the PO is compressed.

Hierarchical Representation

3 objectInformation (1-*: M, R)

3.4 objectPackagingFormat (1-1, M, NR)

3.4.1 objectPackagingFormatName (1-1: M, NR)

7.5 Fixity Category

Fixity is used to demonstrate that the particular content information has not been altered in an undocumented or unauthorized manner.

The elements of the Fixity category are:

— **Last fixity check date (lastCheckDate) —** the date of the last fixity check according to ISO 8601:2004, Data elements and interchange formats — Information interchange — Representation of dates and times, e.g., YYYY-MM-DDThh:mm:ss.ssssssZ.

— **Digest information (digestInformation) —** information about the multiple fixity checks.

In some systems, the fixity check date is auditable, but this document does not mandate that in SIRF as it requires a significant overhead, and not all systems support that.

The digestInformation element is composed of the following attributes:

— **Fixity Originator (digestOriginator) —** the agent that created the original digest that is compared in a fixity check.

— **Fixity algorithm (digestAlgorithm) —** the specific algorithm used to construct the digest value. Examples include MD5, SHA-256, Whirlpool.

— **Fixity value (digestValue) —** the output of the fixity algorithm which is stored so that it can be compared in future fixity checks. For composite object — the value is the fixity of the manifest.

Hierarchical Representation

There may be multiple triplets: {digestOriginator, digestAlgorithm, digestValue}.

The hierarchical representation of the Fixity category is as follows:

3 objectInformation (1-*: M, R)

3.5 objectFixity (1-1: M, NR)

3.5.1 lastCheckDate (1-1: M, NR)

3.5.2 digestInformation(1-*: M, R)

3.5.2.1 digestOriginator (1-1: M, NR)

3.5.2.2 digestAlgorithm (1-1: M, NR)

3.5.2.3 digestValue (1-1: M, NR)

7.6 Retention Category

A storage system implementing SIRF may optionally implement retention management disciplines into the system management functionality. Retention management includes implementing a retention policy, defining a hold policy to enable objects to be held for specific purposes (e.g., litigation), and defining how the rules for deleting objects are affected by placing either a retention policy and/or a hold on an

object. Preservation object deletion is not a capability of retention management, per se, but rather is a general system capability. However, preservation object hold describes what happens when placing either a retention policy and/or a hold on an object.

When a SIRF object's retention metadata is set, it specifies the time period during which object deletion shall be prohibited. Objects may have multiple retention periods, however, deletion should be prohibited if any of those retention periods are active.

Retention metadata may also include holds. A hold enforces read-only data object access and prohibition of object deletion. Objects may have multiple holds, and while any hold is active, the system shall enforce read-only access to the object and prevent deletion.

While SIRF does provide support for basic retention metadata, the implementation will be dependent on the underlying system. For example, a system implemented on CDMI may take advantage of CDMI's retention mechanisms.

The hierarchical representation of the retention category follows:

Hierarchical Representation

3 objectInformation (1-*: M, R)

3.6 objectRetention (1-1: O, R)

3.6.1 retentionType(1-1:M,NR)

3.6.2 retentionValue(1-1:M,NR)

retentionType can be *time_period*, with the retentionValue specifying the period during which the object must be retained, or retentionType can be *hold* where the value is ignored. All retention holds shall be removed before any modification or deletion of the object is allowed.

7.7 Audit Log Category

The audit log is provided as a place for preserving any important information about how an object has been accessed or modified. Note that this category is for the audit log of a preservation object, in contrast to the container audit log described in the Container Information Metadata section. The extent and contents of an audit log depend on the needs of the specific preservation data store and its use case. Distinct domains have different audit logs regulations e.g., SEC is for the market domain, FDA is for the pharmatutical domain. In SIRF, audit logs are stored as SIRF preservation objects. Therefore, the object's log information is stored as an object ID in the SIRF catalog. SNIA published a whitepaper on audit logging for storage that is available at: http://www.snia.org/sites/default/files/SNIA-Logging-WP.050921.pdf.

This category includes just one element:

— **Audit Log Reference** (objectAuditLogReference) — reference to the object that contains the audit log

The containerAuditLogReference element is composed of the following attributes:

— **Reference Type (referenceType) —** whether internal referencing within the container or external referencing

— Reference Role (referenceRole) — the value should be "AuditLog"

— **Reference Value (referenceValue) —** the unique identifier of the referenced audit log object. If the referenced object is an internal preservation object, the value will be its objectVersionIdentifier. objectIdentifierValue

Hierarchical Representation

3 objectInformation (1-1: M, NR)

3.7 objectAuditLog (0-*, O, R)

3.7.1 objectAuditLogReference (1-1: M, NR)

3.7.1.1 referenceType (1-1: M, NR)

3.7.1.2 referenceRole (1-1: M, NR)

3.7.1.3 referenceValue (1-1: M, NR)

7.8 Extension Category

The extension category is a placeholder for data store-specific information. Each organization using SIRF may use this reserved, general purpose category to add private information or metadata that is specific to their own domain or data store. The information within this category shall be self-describing. The extension is comprised of a description, the name of the organization and a set of key-value pairs. This category is not mandatory.

Hierarchical Representation

3 objectInformation (1-*: M, R)

3.9 objectExtension(0-*, O, R)

3.9.1 objectExtensionPair(0-*, O, R)

3.9.1.1 objectExtensionKey(1-1, M, NR)

3.9.1.2 objectExtensionValue(1-1, M, NR)

3.9.2 objectExtensionOrganization(1-1, M, NR)

3.9.3 objectExtensionDescription(0-1, O, NR)

8 Serialization for SNIA CDMI

The Cloud Data Management Interface (CDMI)[7] is an ISO/IEC 17826:2012 standard that defines an interoperable format for moving data and associated metadata between cloud providers. CDMI defines a RESTful interface where data objects can be accessed by standard browsers and internet tools (subject to owner's access control lists). CDMI data objects may "order" data services from the cloud via Data System Metadata (key/value) on the containers or objects.

CDMI has several implementations including an open source implementation on top of OpenStack Swift.

To enable a CDMI container to be qualified as a SIRF container, it is necessary to add an Extension specification to CDMI that defines a new field for the capabilities object. This new field will say whether the cloud supports making a container SIRF-compliant and will point to the SIRF specification.

A CDMI cloud container can be qualified as a SIRF container when:

— The SIRF magic object is mapped to the CDMI container metadata and includes, for example, specification ID and version, SIRF level, SIRF catalog object ID.

— The SIRF catalog is an object in the CDMI container formatted in JSON (self-describing) that includes one *containerInformation* section and multiple *ObjectInformation* sections — one for each PO within

the container (self-contained). This object should probably be indexed. There is a CDMI extension to support indexing that its granularity is per object.

— A SIRF preservation object (PO) that is a simple object (contains one element) is mapped to a CDMI data object. The simple object can be for example a tar/zip.

— A SIRF PO that is a composite object (contains several elements) is mapped to a set of data objects (one for each element) and a manifest data object that its content includes information about the elements.

The interface to the SIRF-compliant CDMI container is the regular CDMI interface. In addition, the CDMI API can be used to store and access the various preservation objects and the catalog object. In the interface, CDMI content type is used when you need to create/update metadata e.g., updating container metadata with the fields coming from the magic object. Otherwise, you can use non CDMI content type.

For example, assume there is a CDMI container named "Patient Container," (as shown in Figure 3) that is SIRF-compliant and includes medical encounters and images for the patient. Assume each encounter is a simple preservation object; each image is a composite preservation object; and since the container is SIRF-compliant, it also includes a catalog object.

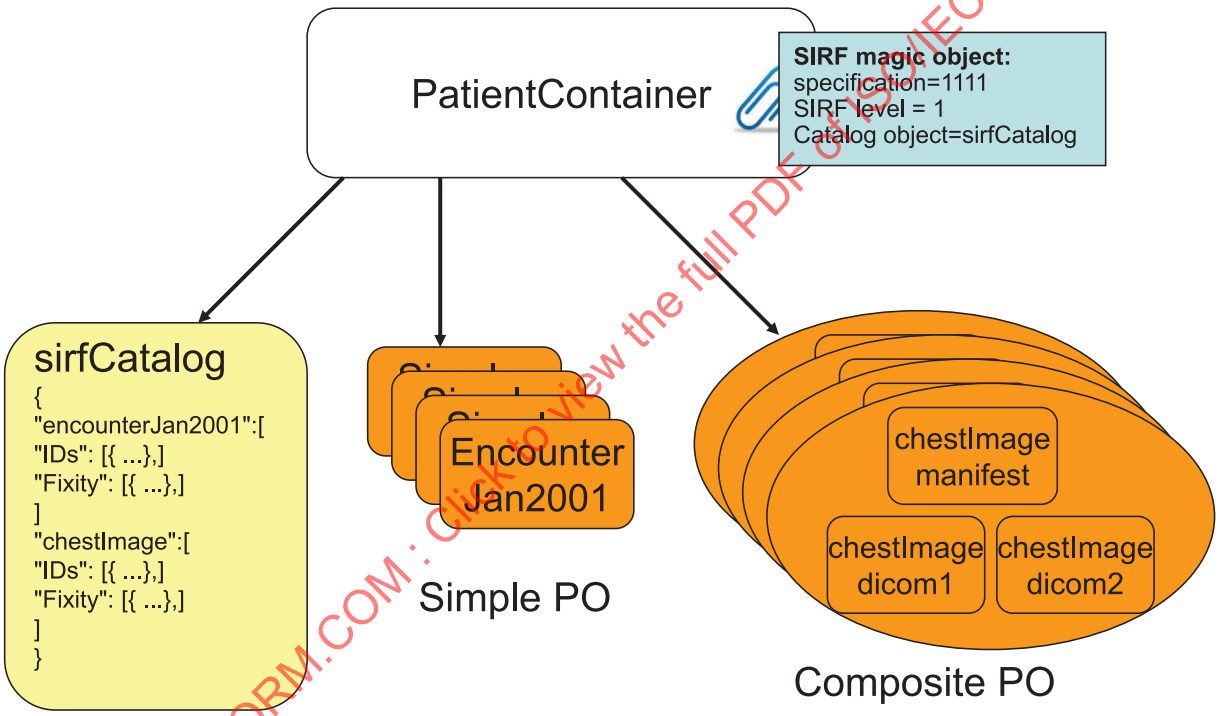

Figure 3 — SIRF Serialization for CDMI Example

One can read the various preservation objects and the catalog object via CDMI REST API as follows:

GET <root URI>/<PatientContainer>/encounterJan2001

GET <root URI>/<PatientContainer>/chestImage

GET <root URI>/<PatientContainer>/sirfCatalog

8.1 Catalog Serialization: Object IDs Category

The Object IDs category in the SIRF catalog is serialized as follows:

— PO name (objectName) — for simple objects, the value is the CDMI object name that corresponds to this PO. For composite objects, the value is the name of the manifest data object. Note that object names are optional in CDMI and also in the SIRF catalog.

— PO version ID (objectVersionIdentifier) — the value is "CDMI" + CDMI container name/ID + CDMI object name/ID. Object IDs are optional in CDMI but then there is a name. Note that a container may be renamed which can cause inconsistency.

— PO logical ID (objectLogicalIdentifier) — the value is the PO version ID of the first version of this PO.

— PO parent ID (objectParentIdentifier) — the value is the PO version ID of the parent PO.

The optional CDMI Versioning Extension specifies interface to manage versioned objects. This is orthogonal to the metadata stored in the SIRF catalog object. A PO version is different than cloud object version e.g., for composite PO.

8.2 Catalog Serialization: Fixity Category

The Fixity category in the SIRF catalog is serialized as follows:

— Last fixity check date — the value is according to ISO 8601:2004 that is also used in CDMI dates.

— Fixity Originator (digestOriginator).

— Fixity algorithm (digestAlgorithm) — the value can be the CDMI cdmi_value_hash data system metadata expressed as a string in the form of ALGORITHM LENGTH e.g., SHA160, SHA256.

— Fixity value (digestValue) — the value can be the CDMI cdmi_hash storage system metadata that represents the hash of the value of the object, encoded using Base16 encoding rules described in RFC 4648.

9 Serialization for SNIA LTFS

The Linear Tape File System (LTFS) Format Specification[8] defines LTFS Volumes. An LTFS Volume holds data files and corresponding metadata to completely describe the directory and file structures stored on the volume. Files can be written to, and read from, an LTFS Volume using standard POSIX file operations. The LTFS Volume includes an index in XML that contains metadata similar to information in disk-based file systems such as file name, dates, extent pointers, extended attributes, etc. LTFS is becoming the standard for linear tape and is being formalized through SNIA.

An LTFS volume is comprised of a pair of LTFS partitions: a data partition (DP) and an index partition (IP), as shown in Figure 4. Each partition contains a Label Construct followed by a Content Area.

— The Label Construct consists of an ANSI VOL1 label, followed by a single file mark, followed by one record in LTFS Label format, followed by a single file mark. The VOL1 Label includes, for example, volume identifier (6 bytes), implementation identifier (13 bytes), owner identifier (14 bytes). The LTFS Label includes, for example, creator, volume UUID, blocksize, compression, partitions ids. Each Label construct for an LTFS volume contains identical information except for the *Location* field of the LTFS Label.

The Content Area contains zero or more Data Extents and Index Constructs in any order. The last construct in the Content Area of a complete partition is an Index Construct. File marks cannot appear anywhere else on the tape except after the labels and around the indexes.

A LTFS tape container can be qualified also as a SIRF container when the volume format is as follows:

— The SIRF magic object is mapped to extended attributes of the “LTFS index” root directory. The magic object includes, for example, specification ID and version, SIRF level, reference to SIRF catalog.

— The SIRF catalog resides in the index partition and formatted in XML (self-describing) that includes one *containerInformation* section and multiple *objectInformation* sections — one for each PO within the container (self-contained). LTFS application has rules to indicate what to store in the index partition. That method can be used to indicate to store the SIRF catalog in the index partition. Alternatively, the index partition can include a reference to the SIRF catalog that will reside in the data partition.

— A preservation object (PO) is mapped to an LTFS file or set of files. In case the PO is a simple object composed of one element, it is mapped to a LTFS file. In case the PO is a composite object composed of several elements, it is mapped to a set of LTFS files (one for each element) and a manifest file that its content includes information about the elements.

— Generations of the SIRF catalog are not maintained; versions of the POs are maintained by the IDs category.

The size of the Index Partition is 2 wraps, which is 37.5GB in LTO5, and it will be larger on LTO6. The LTFS index itself will probably never be larger than a fraction of a GB, so space remains available for the SIRF catalog, and even for additional information e.g., thumbnails of images.

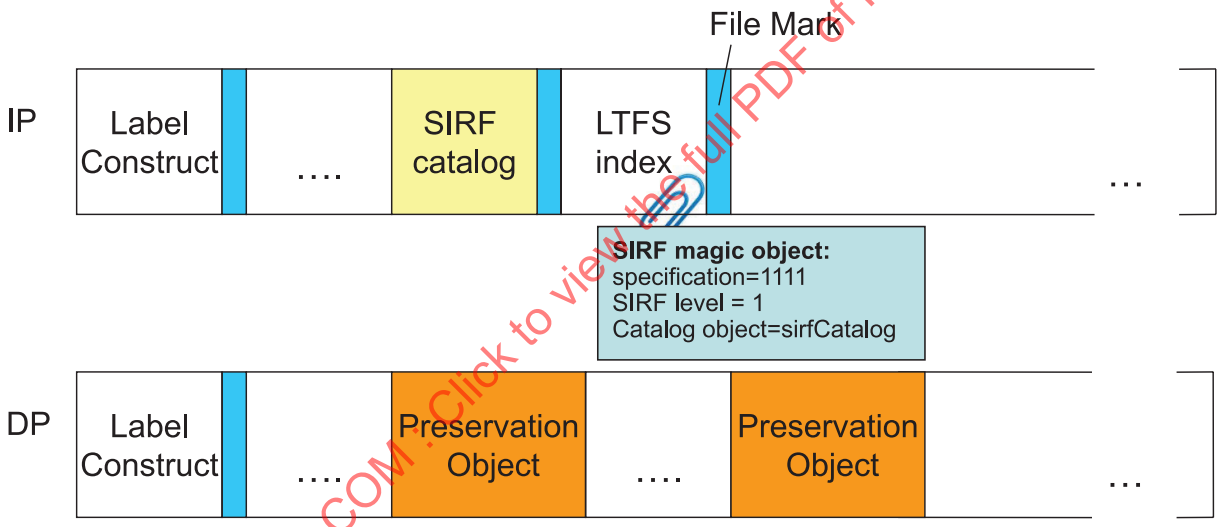

Figure 4 — SIRF Serialization for LTFS Volume

9.1 Catalog Serialization: Object IDs Category

The Object IDs category in the SIRF catalog is serialized as follows:

— PO name (objectName) — for simple objects, the value is the file name that corresponds to this PO. For composite objects, the value is the name of the manifest file.

— PO version ID (objectVersionIdentifier) — the value is "LTO” + Tape UUID + LTFS <fileuid>. The LTFS fileuid is a 64-bit sequence number generated for each LTFS file.

— PO logical ID (objectLogicalIdentifier) — the value is the PO version ID of the first version of this PO.

— PO parent ID (objectParentIdentifier) — the value is the PO version ID of the parent PO.

9.2 Catalog Serialization: Fixity Category

The Fixity category in the SIRF catalog is serialized as follows:

— Last fixity check date — the value is according to ISO 8601:2004 that is also used in LTFS dates.

— Fixity Originator (digestOriginator).

— Fixity algorithm (digestAlgorithm) — a string for the algorithm name e.g., SHA160, SHA256.

— Fixity value (digestValue) — for composite object it includes the fixity of the manifest file. Then, the manifest includes the fixities of the elements.

10 Serialization for OpenStack Swift

OpenStack Swift[9] is a highly available, distributed, eventually consistent object/blob store in the cloud. Swift uses a RESTful interface (HTTP commands, e.g., GET, PUT, POST, DELETE and HEAD) for the creation, manipulation and deletion of storage objects and metadata.

The serialization for Swift involves mapping SIRF objects to Swift Objects. SIRF containers are mapped to Swift containers. The magic object is included in the Swift container's metadata. The catalog is mapped to a file in JSON format[10] (ECMA-404 The JSON Data Interchange Standard) in the container (its name is specified in the magic object; catalog.json is the default). The preservation objects are mapped to Swift objects in the container and referenced in the catalog.

The retrieval of information is similar to the CDMI serialization:

GET <root URI>/<Container>/<PO name>

GET <root URI>/<Container>/<Catalog file name>

GET <root URI>/<Container>/provenance.po.json (provenance preservation object)

Container provenance is one of the POs (provenance.po.json). Therefore, the catalog references the container provenance twice: in the container specification (in the container information section), and as a regular PO.

Below are some examples of using Swift's RESTful API to manipulate SIRF objects. Our example container only contains one preservation object, which is the provenance information.

1. **Retrieving container metadata (HTTP HEAD is used)**

```
HEAD http://$SWIFT_IP:$SWIFT_PORT/v1/$SWIFT_ACCOUNT/ContainerExample

HTTP/1.1 204 No Content
Content-Length: 0
X-Container-Object-Count: 3
X-Container-Meta-Sirfcatalogid: catalog.json
Accept-Ranges: bytes
X-Storage-Policy: Policy-0
X-Container-Meta-Sirflevel: 1
X-Container-Meta-Containerspecification: 1.0
X-Container-Bytes-Used: 4039
X-Timestamp: 1416832358.29789
Content-Type: text/plain; charset=utf-8
X-Trans-Id: txfa2c6026e1ea43c7a440f-0054734cdb
Date: Mon, 24 Nov 2014 15:20:59 GMT
```

The lines in bold correspond to the magic object which is embedded in the header of the HTTP request.

2. **Listing the files in the container named 'ContainerExample' (HTTP GET is used)**

```
GET http://$SWIFT_IP:$SWIFT_PORT/v1/$SWIFT_ACCOUNT/ContainerExample

HTTP/1.1 200 OK
Content-Length: 65
X-Container-Object-Count: 3
X-Container-Meta-Sirfcatalogid: catalog.json
Accept-Ranges: bytes
X-Storage-Policy: Policy-0
X-Container-Meta-Sirflevel: 1
X-Container-Meta-Containerspecification: 1.0
X-Container-Bytes-Used: 4039
X-Timestamp: 1416832358.29789
Content-Type: text/plain; charset=utf-8
X-Trans-Id: tx6642b04344064a029fcc2-0054734d08
Date: Mon, 24 Nov 2014 15:21:44 GMT

ContainerExample-A-Valentine-1.0
catalog.json
provenance.po.json
```

The lines in bold correspond to the Swift objects in the container. Please note that the container contains the catalog with the same name specified in the magic object, as well as two POs called provenance.po.json (which corresponds to the provenance information) and ContainerExample-A-Valentine-1.0.

3. Retrieving a PO (HTTP GET is used)

```
GET http://$SWIFT_IP:$SWIFT_PORT/v1/$SWIFT_ACCOUNT/ContainerExample/ContainerExample-A-Valentine-1.0

HTTP/1.1 200 OK
Content-Length: 990
Accept-Ranges: bytes
Last-Modified: Mon, 24 Nov 2014 15:20:01 GMT
Etag: 8d766d52bdf6d9d18721070f218b29ca
X-Timestamp: 1416842400.31717
Content-Type: application/unknown
X-Trans-Id: tx69de11dc395544d8bf61a-0054734d31
Date: Mon, 24 Nov 2014 15:22:25 GMT

For her this rhyme is penned, whose luminous eyes,
Brightly expressive as the twins of Lœda,
Shall find her own sweet name, that, nestling lies
Upon the page, enwrapped from every reader.
Search narrowly the lines! -- they hold a treasure
Divine -- a talisman -- an amulet
That must be worn at heart. Search well the measure --
The words -- the syllables! Do not forget
The trivialest point, or you may lose your labor!
And yet there is in this no Gordian knot
Which one might not undo without a sabre,
If one could merely comprehend the plot.
Enwritten upon the leaf where now are peering
Eyes scintillating soul, there lie perdus
Three eloquent words oft uttered in the hearing
Of poets, by poets -- as the name is a poet's, too.
Its letters, although naturally lying
Like the knight Pinto -- Mendez Ferdinando --
Still form a synonym for Truth. -- Cease trying!
You will not read the riddle, though you do the best you can do.

Edgar Allan Poe
```

The lines in bold correspond to the contents of the preservation object.

4. Retrieving the catalog (HTTP GET is used)

```
GET http://$SWIFT_IP:$SWIFT_PORT/v1/$SWIFT_ACCOUNT/ContainerExample/catalog.json


HTTP/1.1 200 OK
Content-Length: 3066
Accept-Ranges: bytes
Last-Modified: Fri, 23 Jan 2015 15:02:39 GMT
Etag: c520a08ed64a5bc714ad2712eb62ed2c
X-Timestamp: 1422025358.98079
Content-Type: application/unknown
X-Object-Meta-Key1: value1
X-Trans-Id: txf2c6ef4e38624a0684b39-0054c2629b
Date: Fri, 23 Jan 2015 15:02:51 GMT


{"catalogId":"catalog.json","containerInformation":{"containerSpecification":{"container
SpecificationIdentifier":"SIRF-1.0","containerSpecificationSirfLevel":"1","containerSpecifi
cationVersion":"1.0"},"containerIdentifier":{"containerIdentifierLocale":"en","containerI
dentifierType":"containerIdentifier","containerIdentifierValue":"ContainerExample"},"conta
inerState":{"containerStateType":"ready","containerStateValue":"true"},"containerProven
anceReference":{"referenceRole":"Provenance","referenceType":"internal","referenceValue
":"http://snia.org/sirf/ContainerExample-provenance.po.json-1.0"},"containerAuditLog":[
]},"objectsSet":{"objectInformation":[{"objectIdentifiers":[{"objectName":[{"objectIdent
ifierLocale":"en","objectIdentifierType":"name","objectIdentifierValue":"provenance.po.jso
n"}],"objectLogicalIdentifier":{"objectIdentifierLocale":"en","objectIdentifierType":"logi
calIdentifier","objectIdentifierValue":"http://snia.org/sirf/ContainerExample-provenance.
po.json"},"objectParentIdentifier":{"objectIdentifierLocale":"en","objectIdentifierType":"
parentIdentifier","objectIdentifierValue":"null"},"objectVersionIdentifier":{"objectIdentif
ierLocale":"en","objectIdentifierType":"versionIdentifier","objectIdentifierValue":"http://
snia.org/sirf/ContainerExample-provenance.po.json-1.0"}}],"objectCreationDate":"2015-01-
23T13:02Z","objectLastModifiedDate":"2015-01-23T13:02Z","objectLastAccessedDate":"2015-01-
23T13:02Z","objectRelatedObjects":[],"packagingFormat":{"packagingFormatName":"none"},"o
bjectFixity":{"digestInformation":[{"digestAlgorithm":"SHA-1","digestOriginator":"Object
Api","digestValue":"7bec3092783ac1ffe4ff4b0c98958e2b776a4e2"}],"lastCheckDate":"201501
23130218"},"objectRetention":{"retentionType":"time_period","retentionValue":"forever"},
"objectAuditLog":[],"objectExtension":[]},{"objectIdentifiers":[{"objectName":[{"objectI
dentifierLocale":"en","objectIdentifierType":"name","objectIdentifierValue":"A-Valentine"}]
,"objectLogicalIdentifier":{"objectIdentifierLocale":"en","objectIdentifierType":"logicalI
dentifier","objectIdentifierValue":"ContainerExample-A-Valentine"},"objectParentIdentifier"
:{"objectIdentifierLocale":"en","objectIdentifierType":"parentIdentifier","objectIdentifier
Value":"null"},"objectVersionIdentifier":{"objectIdentifierLocale":"en","objectIdentifierTy
pe":"versionIdentifier","objectIdentifierValue":"ContainerExample-A-Valentine-1.0"}}],"ob
jectCreationDate":"2015-01-23T13:02Z","objectLastModifiedDate":"2015-01-23T13:02Z","obj
ectLastAccessedDate":"2015-01-23T13:02Z","objectRelatedObjects":[],"packagingFormat":{
"packagingFormatName":"none"},"objectFixity":{"digestInformation":[{"digestAlgorithm":
"SHA-1","digestOriginator":"ObjectApi","digestValue":"307257C086B7F24759E2E58D65313ED5
FBC7D6FC"}],"lastCheckDate":"20150123130256"},"objectRetention":{"retentionType":"time_
period","retentionValue":"10 years"},"objectAuditLog":[],"objectExtension":[]}]}}
```


11 Use Case Example

UC8: BioMedical Bank, from the *SIRF use cases and functional requirements, working draft — version 0.5a* is used here to demonstrate the identifiers in the SIRF catalog. The following actors of a preservation system are defined in the *SIRF use cases and functional requirements, working draft — version 0.5a* and depicted in Figure 5:

— *Storage* — Storage subsystem that persists numerous preservation objects.

— *TP-Service* — Today's preservation service, e.g., OAIS ingest service, transformation service.

— *FP-Service* — Future preservation service, which may be unknown today.

— *T-App* — Today's application that generates digital data, e.g., a word processor, eMail application.

— *F-App* — Future application, which may be unknown today.

— *Reg* — Registry that stores representation information of the used storage formats, e.g., the specification documents of the used formats.

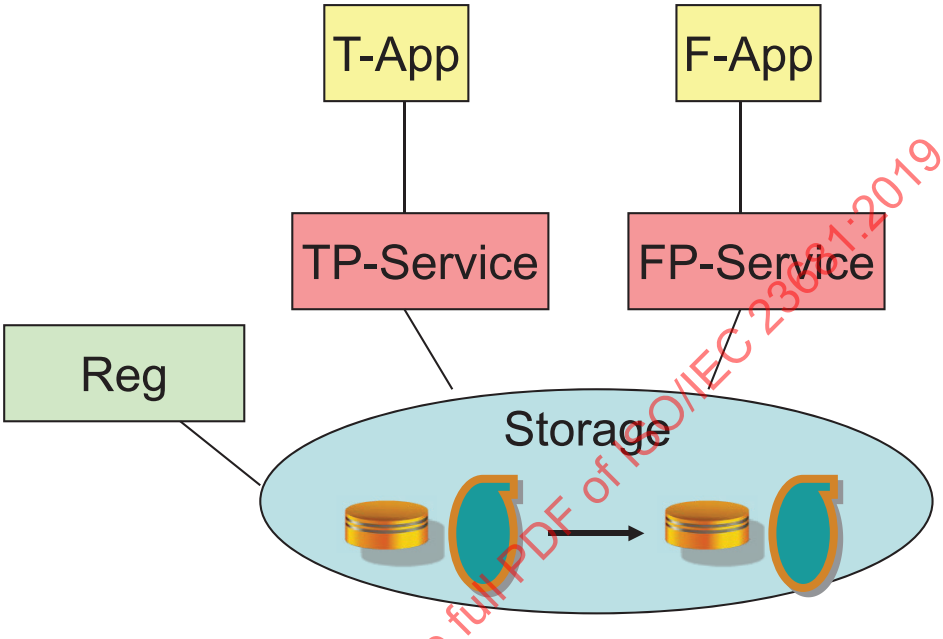

Figure 5 — SIRF Actors

The scenario in this use case is as follows: A large hospital, which also has an adjacent academic medical research center, stores the patients' biomedical data in a biomedical bank, in which data is preserved for decades. The data is used at the point of care as well as for biomedical research by the adjacent research center.

Table 2 describes the use case with its associated information in the SIRF container.

Table 2 — BioMedical Bank Example

	Use Case Flow	SIRF Container
1.	T-App ingests via TP-service a PO that includes a standardized Digital Imaging and Communications in Medicine (DICOM) image of the leg of a patient that is a minor.	PO_1 for the image is stored in the SIRF container, and the catalog includes an entry *objectInformation* for this PO_1 with elements *objectName*, *objectVersionIdentifier*, *objectLogicalIdentifier*.
2.	Time passes and the patient, who is now an adult, schedules an appointment to check a new problem he has encountered in his leg.	During that time PO_1 may be subject to transformations for logical preservation. For example PO_2 may be created in the SIRF container and the catalog will include an entry objectInformation for this PO_2 with elements *objectName*, *objectVersionIdentifier*, *objectLogicalIdentifier* that is identical to *objectLogicalIdentifier* of PO_1 and *objecParentIdentifier* that is identical to *objectVersionIdentifier* of PO_1.
3.	FP-service will identify the data needed for the scheduled appointment using reference, context and provenance information.	The SIRF catalog that may have an online version is searched to find the PO_1 and PO_2 information.

Table 2 *(continued)*

	Use Case Flow	SIRF Container
4.	The identified Preservation Objects will be a-priory brought from an offline media to an online media to be timely accessible for the appointment.	SIRF container is accessed to download PO_1 and PO_2 The POs provenance and authenticity can be verified.
5.	F-App at point of care accesses the identified POs for the patient via FP-Service.	This activity is in a higher layer and does not involve the SIRF container.
6.	More time passes and a researcher from the adjacent academic medical research center wants to access that image for research purposes. According to HIPAA regulations, the researcher can get just a de-identified image.	The SIRF container includes a PO_{100} for the de-identification module, and the SIRF catalog includes an entry *objectInformation* for this PO_{100}. SIRF container is accessed to execute the de-identification module on PO_1 and PO_2
7.	F-App accesses the de-identified PO via FP-Service.	This activity is in a higher layer and does not involve the SIRF container.

Annex A
(informative)

XML schema for the SIRF catalog

This annex contains an XML schema that can be used to verify that an XML file is a SIRF catalog.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="sirfCatalog">
  <xs:complexType>
   <xs:sequence>
    <xs:element type="xs:string" name="catalogId"/>
    <xs:element name="containerInformation" maxOccurs="1" minOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="containerSpecification" maxOccurs="1" minOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element type="xs:string" name="containerSpecificationIdentifier"/>
          <xs:element type="xs:byte" name="containerSpecificationSirfLevel"/>
          <xs:element type="xs:float" name="containerSpecificationVersion"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="containerIdentifier" maxOccurs="1" minOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element type="xs:string" name="containerIdentifierLocale" maxOccurs="1"
minOccurs="1"/>
          <xs:element type="xs:string" name="containerIdentifierType" maxOccurs="1"
minOccurs="1"/>
          <xs:element type="xs:string" name="containerIdentifierValue" maxOccurs="1"
minOccurs="1"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="containerState" maxOccurs="1" minOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element type="xs:string" name="containerStateType" maxOccurs="1"
minOccurs="1"/>
          <xs:element type="xs:string" name="containerStateValue" maxOccurs="1"
minOccurs="1"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="containerProvenanceReference" maxOccurs="1" minOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element type="xs:string" name="referenceRole" maxOccurs="1" minOccurs="1"/>
          <xs:element type="xs:string" name="referenceType" maxOccurs="1" minOccurs="1"/>
          <xs:element type="xs:anyURI" name="referenceValue" maxOccurs="1" minOccurs="1"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="containerAuditLog" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="containerAuditLogReference" maxOccurs="1" minOccurs="1">
           <xs:complexType>
            <xs:sequence>
             <xs:element type="xs:string" name="referenceRole" maxOccurs="1"
minOccurs="1"/>
             <xs:element type="xs:string" name="referenceType" maxOccurs="1"
```

```
minOccurs="1"/>
             <xs:element type="xs:string" name="referenceValue" maxOccurs="1"
minOccurs="1"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="objectsSet" maxOccurs="1" minOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="objectInformation" maxOccurs="unbounded" minOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="objectIdentifiers" maxOccurs="unbounded" minOccurs="1">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="objectName" maxOccurs="unbounded" minOccurs="0">
              <xs:complexType>
               <xs:sequence>
                <xs:element type="xs:string" name="objectIdentifierLocale" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierType" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="objectLogicalIdentifier" maxOccurs="1" minOccurs="1">
              <xs:complexType>
               <xs:sequence>
                <xs:element type="xs:string" name="objectIdentifierLocale" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierType" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="objectParentIdentifier" maxOccurs="1" minOccurs="0">
              <xs:complexType>
               <xs:sequence>
                <xs:element type="xs:string" name="objectIdentifierLocale" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierType" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="objectVersionIdentifier" maxOccurs="unbounded"
minOccurs="1">
              <xs:complexType>
               <xs:sequence>
                <xs:element type="xs:string" name="objectIdentifierLocale" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierType" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectIdentifierValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
```

```
            </xs:complexType>
           </xs:element>
           <xs:element type="xs:string" name="objectCreationDate" maxOccurs="1"
minOccurs="1"/>
           <xs:element type="xs:string" name="objectLastModifiedDate" maxOccurs="1"
minOccurs="0"/>
           <xs:element type="xs:string" name="objectLastAccessedDate" maxOccurs="1"
minOccurs="0"/>
           <xs:element name="objectRelatedObjects" maxOccurs="unbounded" minOccurs="0">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="objectRelatedObjectsReference" maxOccurs="1" minOccurs="1">
               <xs:complexType>
                <xs:sequence>
                 <xs:element type="xs:string" name="referenceRole" maxOccurs="1"
minOccurs="1"/>
                 <xs:element type="xs:string" name="referenceType" maxOccurs="1"
minOccurs="1"/>
                 <xs:element type="xs:string" name="referenceValue" maxOccurs="1"
minOccurs="1"/>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="packagingFormat" maxOccurs="1" minOccurs="1">
            <xs:complexType>
             <xs:sequence>
              <xs:element type="xs:string" name="packagingFormatName" maxOccurs="1"
minOccurs="1"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="objectFixity" maxOccurs="1" minOccurs="1">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="digestInformation" maxOccurs="unbounded" minOccurs="1">
               <xs:complexType>
                <xs:sequence>
                 <xs:element type="xs:string" name="digestAlgorithm" maxOccurs="1"
minOccurs="1"/>
                 <xs:element type="xs:string" name="digestOriginator" maxOccurs="1"
minOccurs="1"/>
                 <xs:element type="xs:string" name="digestValue" maxOccurs="1"
minOccurs="1"/>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
              <xs:element type="xs:long" name="lastCheckDate" maxOccurs="1" minOccurs="1"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="objectRetention" maxOccurs="1" minOccurs="1">
            <xs:complexType>
             <xs:sequence>
              <xs:element type="xs:string" name="retentionType" maxOccurs="1"
minOccurs="1"/>
              <xs:element type="xs:string" name="retentionValue" maxOccurs="1"
minOccurs="1"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="objectAuditLog" maxOccurs="unbounded" minOccurs="0">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="objectAuditLogReference" maxOccurs="1" minOccurs="1">
               <xs:complexType>
                <xs:sequence>
                 <xs:element type="xs:string" name="referenceRole" maxOccurs="1"
minOccurs="1"/>
```

```
                <xs:element type="xs:string" name="referenceType" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="referenceValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="objectExtension" maxOccurs="unbounded" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="objectExtensionPair" maxOccurs="unbounded" minOccurs="0">
              <xs:complexType>
               <xs:sequence>
                <xs:element type="xs:string" name="objectExtensionKey" maxOccurs="1"
minOccurs="1"/>
                <xs:element type="xs:string" name="objectExtensionValue" maxOccurs="1"
minOccurs="1"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element type="xs:string" name="objectExtensionDescription" maxOccurs="1"
minOccurs="0"/>
             <xs:element type="xs:string" name="objectExtensionOrganization" maxOccurs="1"
minOccurs="1"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Annex B
(informative)

Sample XML catalog

This annex contains an example of a SIRF catalog in XML format. This annex shows the various SIRF elements described in this document as they would appear in a production system.

```
<sirfCatalog>
 <catalogId>catalog.xml</catalogId>
 <containerInformation>
  <containerSpecification>
   <containerSpecificationIdentifier>SIRF-1.0</containerSpecificationIdentifier>
   <containerSpecificationSirfLevel>1</containerSpecificationSirfLevel>
   <containerSpecificationVersion>1.0</containerSpecificationVersion>
  </containerSpecification>
  <containerIdentifier>
   <containerIdentifierLocale>en</containerIdentifierLocale>
   <containerIdentifierType>containerIdentifier</containerIdentifierType>
   <containerIdentifierValue>SampleContainer</containerIdentifierValue>
  </containerIdentifier>
  <containerState>
   <containerStateType>ready</containerStateType>
   <containerStateValue>true</containerStateValue>
  </containerState>
  <containerProvenanceReference>
   <referenceRole>Provenance</referenceRole>
   <referenceType>internal</referenceType>
   <referenceValue>http://snia.org/sirf/SampleContainer-provenance.po.json-
1.0</referenceValue>
  </containerProvenanceReference>
  <containerAuditLog>
   <containerAuditLogReference>
    <referenceRole>AuditLog</referenceRole>
    <referenceType>external</referenceType>
    <referenceValue>http://link.to/auditLog</referenceValue>
   </containerAuditLogReference>
  </containerAuditLog>
  <containerAuditLog>
   <containerAuditLogReference>
    <referenceRole>AuditLog</referenceRole>
    <referenceType>internal</referenceType>
    <referenceValue>audit_log_object_version_identifier</referenceValue>
   </containerAuditLogReference>
  </containerAuditLog>
 </containerInformation>
 <objectSet>
  <objectInformation>
   <objectIdentifiers>
    <objectName>
     <objectIdentifierLocale>en</objectIdentifierLocale>
     <objectIdentifierType>name</objectIdentifierType>
     <objectIdentifierValue>provenance.po.json</objectIdentifierValue>
    </objectName>
    <objectLogicalIdentifier>
     <objectIdentifierLocale>en</objectIdentifierLocale>
     <objectIdentifierType>logicalIdentifier</objectIdentifierType>
     <objectIdentifierValue>http://snia.org/sirf/SampleContainer-
provenance.po.json</objectIdentifierValue>
    </objectLogicalIdentifier>
    <objectParentIdentifier>
     <objectIdentifierLocale>en</objectIdentifierLocale>
     <objectIdentifierType>parentIdentifier</objectIdentifierType>
     <objectIdentifierValue>null</objectIdentifierValue>
    </objectParentIdentifier>
```