LTFS Bulk Transfer

Version 1.0

ABSTRACT: The LTFS Bulk Transfer standard defines a method by which a set of files, directories and objects from a source system can be transferred to a destination system. The bulk transfer of large quantities of data is well suited for LTFS due to the economic and environmental characteristics of tape. Building on top of the LTFS format, a standardized method for transferring data is defined that provides many advantages.

SNIA Technical Position

October 11, 2016

USAGE

The SNIA hereby grants permission for individuals to use this document for personal use only, and for corporations and other business entities to use this document for internal use only (including internal copying, distribution, and display) provided that:

1. Any text, diagram, chart, table or definition reproduced shall be reproduced in its entirety with no alteration, and,

2. Any document, printed or electronic, in which material from this document (or any portion hereof) is reproduced shall acknowledge the SNIA copyright on that material, and shall credit the SNIA for granting permission for its reuse.

Other than as explicitly provided above, you may not make any commercial use of this document, sell any or this entire document, or distribute this document to third parties. All rights not explicitly granted are expressly reserved to SNIA.

Permission to use this document for purposes other than those enumerated above may be requested by e-mailing tcmd@snia.org. Please include the identity of the requesting individual and/or company and a brief description of the purpose, nature, and scope of the requested use.

All code fragments, scripts, data tables, and sample code in this SNIA document are made available under the following license:

BSD 3-Clause Software License

Copyright (c) 2016, The Storage Networking Industry Association.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of The Storage Networking Industry Association (SNIA) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Revision History

Date	Version	By	Comments
2013-11-11	1.0a	LTFS TWG	Edits from November Technical Symposium
2014-01-28	1.0b	LTFS TWG	Edits from January SNIA Symposium
2014-03-17	1.0c	LTFS TWG	Edits from March SNIA Symposium
2014-05-13	1.0d	LTFS TWG	Edits from Technical Editor and May SNIA Symposium
2014.05.27	1.0e	LTFS TWG	Added standardized file name, minor corrections to figures, and fixes to XML Schemas
2014-09-02	1.0f	LTFS TWG	Updated name of the document to "LTFS Bulk Transfer"
2016-10-11	1.0	LTFS TWG	Approved by SNIA Membership as a SNIA Technical Position

Contents

1. LTFS BULK TRANSFER 7

1.1 INTRODUCTION 7
1.2 SCOPE 8
1.3 DEFINITIONS AND ACRONYMS 8

2. TRANSFER WORKFLOWS 9

2.1 LTFS TRANSFER TO ENTERPRISE DESTINATION 9
2.2 LTFS TRANSFER TO CLOUD DESTINATION 11
2.3 LTFS TRANSFER FROM CLOUD SOURCE 14
2.4 LTFS TRANSFER BETWEEN CLOUDS 17
2.5 BROKERED LTFS TRANSFER BETWEEN CLOUDS 20

3. TRANSFER FAILURES 23

3.1 INSUFFICIENT PERMISSION ERRORS 23
3.2 DESTINATION WRITE ERRORS 23
3.3 SOURCE READ ERRORS 23
3.4 ITEM MISSING ERRORS 23
3.5 HASH MISMATCH ERRORS 23
3.6 VOLUME MISSING ERRORS 24
3.7 VOLUME DECRYPTION ERRORS 24

4. RECOVERY FROM FAILURES 25

4.1 INSUFFICIENT PERMISSION ERRORS 25
4.2 DESTINATION WRITE ERRORS 25
4.3 SOURCE READ ERRORS 26
4.4 ITEM MISSING ERRORS 26
4.5 HASH MISMATCH ERRORS 26
4.6 VOLUME MISSING ERRORS 26

5. XML DEFINITIONS 27

5.1 TRANSFER REQUEST XML 27
5.1.1 Example Transfer Request XML 27
5.1.2 Transfer Request XML Elements 28
5.2 TRANSFER REPORT XML 32
5.2.1 Example Transfer Report XML 32
5.2.2 Transfer Request XML Elements 33

6. SECURITY CONSIDERATIONS 35

6.1 IDENTIFY VERIFICATION 35
6.2 KEY MANAGEMENT 35
6.3 WEAPONIZED TRANSFERS 35
6.3.1 Deletion 35
6.3.2 Sparse File Expansion 35
6.3.3 Compression Expansion 36

6.3.4 Block Reference Expansion 36
6.4 RECOMMENDATIONS 36
APPENDIX A. TRANSFER REQUEST XML SCHEMA 37
APPENDIX B. TRANSFER REPORT XML SCHEMA 40

1. LTFS Bulk Transfer

1.1 Introduction

The LTFS Bulk Transfer standard defines a method by which a set of files, directories and objects from a source system can be transferred to a destination system.

The bulk transfer of large quantities of data is well suited for LTFS due to the economic and environmental characteristics of tape. Building on top of the LTFS format, a standardized method for transferring data provides the following advantages:

- Provides a uniform way to initiate and accept transfers
- Defines a manifest of which files are to be transferred, which communicates intent to transfer
- Instructs how the files should be merged into the destination namespace
- Facilitates the verification of the integrity and completeness of the transfer
- Defines error handling and recovery behaviors

Together, these capabilities reduce the complexity of transferring data, and allow many aspects of data transfer to be automated.

The LTFS Bulk Transfer standard is specifically targeted for the following use cases:

- An enterprise needs to transfer a large volume of data to or from a remote location
- An enterprise needs to update or synchronize a subset of data stored at a remote location
- An enterprise needs to transfer a large volume of data to a second enterprise, public, or private cloud
- An enterprise needs to transfer a large volume of data from a public or private cloud
- An enterprise needs to transfer a large volume of data from one public or private cloud to a second public or private cloud

Enterprise and cloud use cases are separated due to the extent of automation and self-service facilities typically implemented by cloud providers.

A high-level overview of the LTFS Bulk Transfer process is illustrated in Figure 1.

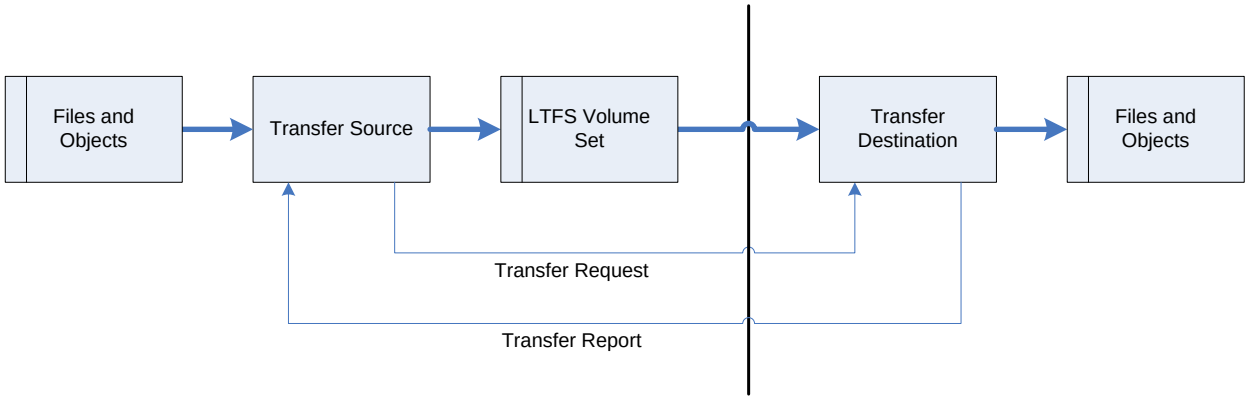

Figure 1 - High-level overview of the LTFS Bulk Transfer process

Here a "Transfer Source" creates an LTFS Volume Set containing files and objects that are desired to be transferred. In addition, an XML-based "Transfer Request" is generated, which is sent to the "Transfer Destination" along with the LTFS Volume Set.

The Transfer Destination processes the Transfer Request and the LTFS Volume Set to merge the specified files and objects into the destination. An XML-based Transfer Report is optionally generated, which can be sent back to the Transfer Source to indicate the results of the transfer process.

1.2 Scope

This document defines the LTFS Bulk Transfer workflows, the LTFS Bulk Transfer XML documents, and provides guidelines for error handling and security.

1.3 Definitions and Acronyms

For the purposes of this document the following definitions and acronyms shall apply.

LTFS – Linear Tape File System

Transfer Initiator – The entity that initiates a transfer operation and receives the transfer results

Transfer Source – The entity originating data in a transfer operation

Transfer Destination – The entity receiving data in a transfer operation

Transfer Operation – The process of selecting, describing, packing, delivering, unpacking, verifying and reporting on the movement of data

Transfer Request XML – A document describing the data to be transferred and the transfer parameters

Transfer Report XML – A document describing the data that was transferred and transfer results

2. Transfer Workflows

2.1 LTFS Transfer to Enterprise Destination

Use cases exist where large quantities of data must be transferred within or between organizations. The LTFS Bulk Transfer standard enables the workflow shown in Figure 2 for the peer-to-peer transfer of data:

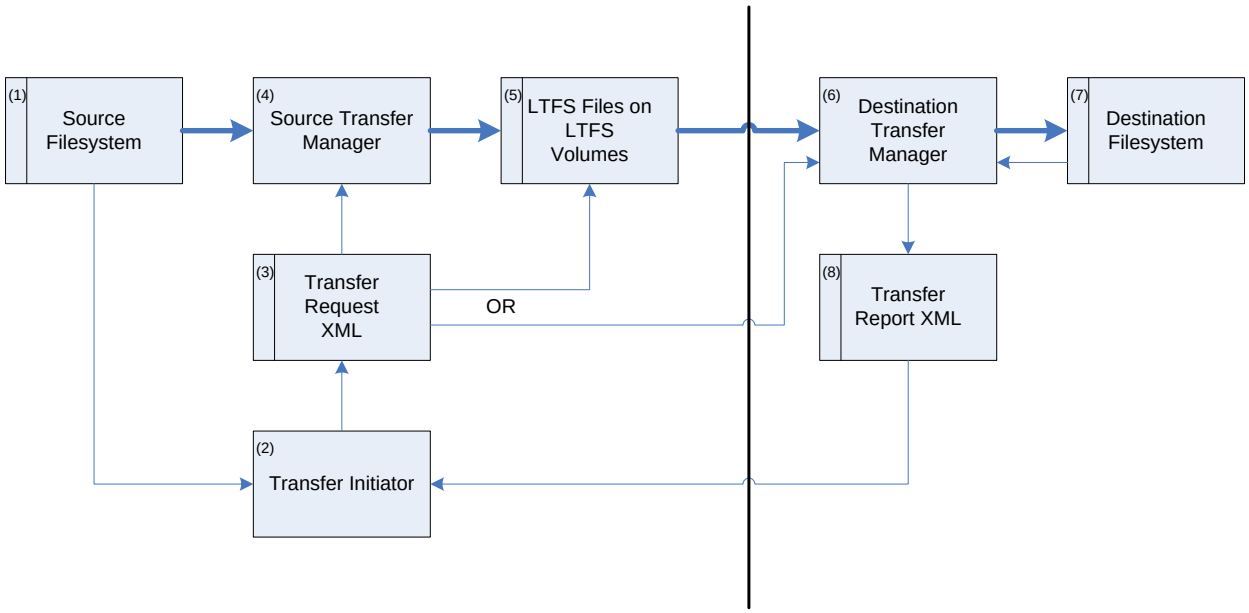

Figure 2 – Peer-to-Peer LTFS Transfer

Table 1 shows the steps performed as part of a peer-to-peer transfer.

Table 1 -Steps in Peer-to-Peer LTFS Transfer

Transfer Step	Implementation Notes
1. A Transfer Initiator (2) desires to transfer files from a local Source Filesystem (1) to a remote Destination Filesystem (7).	

Transfer Step	Implementation Notes
2. The Transfer Initiator (2) selects a subset of files from the Source Filesystem (1), and specifies how the files are to be merged into the Destination Filesystem (7). In accordance with LTFS Bulk Transfer standard, a Transfer Request XML document (3) is generated that specifies the desired transfer.	This functionality can be implemented as a locally run software tool, or through a web-based self-service interface.
3. The Source Transfer Manager (4) takes the Transfer Request XML (3), and creates a set of LTFS Volumes (5) containing the files specified in the XML. If the Transfer Request XML (3) is to be sent in-band, it is also stored as a file named "transfer.xml" in the root directory of each LTFS Volume (5).	This functionality can be implemented as a locally run software tool (and may be integrated with the Transfer Initiator), or can be a non-user visible background software process that automates the collection, packaging and shipment of data. One of the responsibilities of the Source Transfer Manager is to verify that the data is being sent to a legitimate destination. This is out of scope of the LTFS Bulk Transfer standard, and may include functions such as comparison against a whitelist, analysis for loss-prevention, copyright and/or license conformance verification. Departmental charge-back and billing may also be calculated as part of this step.
4. The LTFS Volumes (5) are physically transported to the Destination Transfer Manager (6).	
5. The Transfer Request XML (3) is received by the Destination Transfer Manager (6), either by reading from the LTFS Volumes (5) or out-of-band.	
6. If LTFS Volume encryption is used, the LTFS Volume keys are transported to the Destination Transfer Manager (6) out-of-band.	These keys should be transported over a secure communication channel.

Transfer Step	Implementation Notes
7. The Destination Transfer Manager (6) reads the Transfer Request XML (3), and uses it to verify and merge the files on the LTFS Volumes (5) into the Destination Filesystem (7).	
8. If requested in the Transfer Request XML (3), the Destination Transfer Manager (6) creates a Transfer Report XML document (8) that can be sent back to the Transfer Initiator (2)	
9. The Transfer Initiator (2) receives the Transfer Report XML (8), and can use the included information to signal success or failure to the initiating user, and optionally use the provided information to create a second transfer to allow recovery from various error conditions. (See Section 4 - Recovery from Failures)	

2.2 LTFS Transfer to Cloud Destination

Use cases exist where large quantities of data must be transferred to a public or private cloud. The LTFS Bulk Transfer standard enables the workflow shown in Figure 3 for data transfer to clouds.

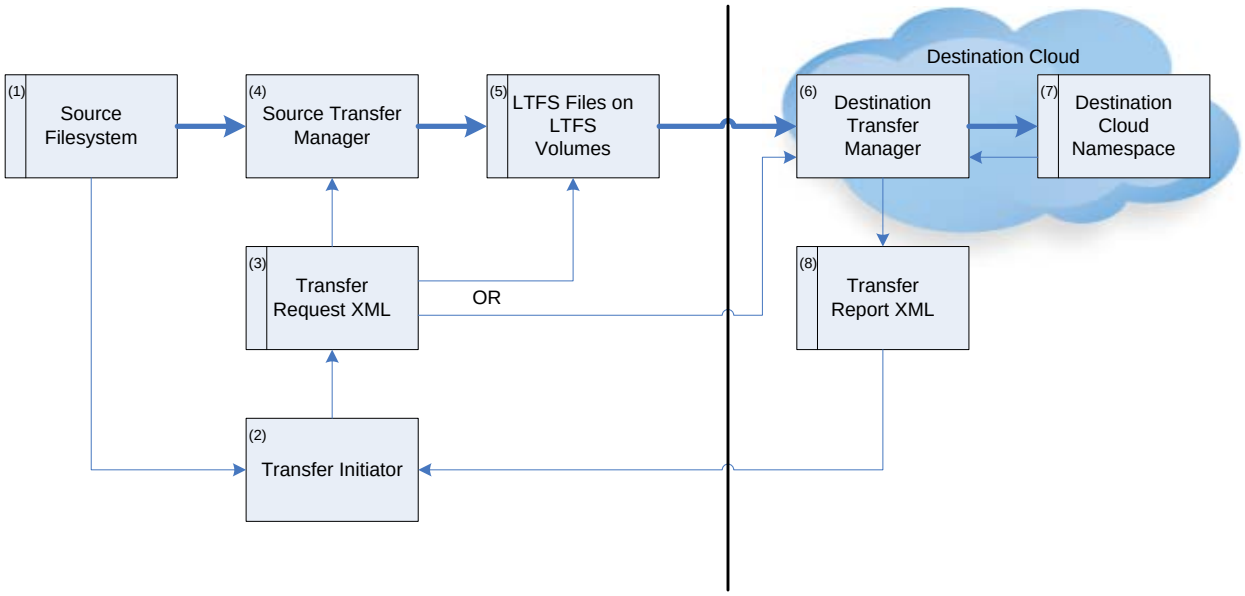

Figure 3 - LTFS Transfer to Cloud

The steps shown in Table 2 are performed as part of transfering data to a cloud:

Table 2 - Steps in LTFS Transfer to Cloud

Transfer Step	Implementation Notes
1. A Transfer Initiator (2) desires to transfer files from a local Source Filesystem (1) to a remote Destination Cloud Namespace (7).	An implementation could consist of a feature in a cloud management interface, which allows a user to start a transfer to the cloud. Such an interface can provide a user with software, instructions and information on transfer costs.
2. The Transfer Initiator (2) selects the subset of files from the Source Filesystem (1), and specifies how the files are to be merged into the Destination Cloud Namespace (7). In accordance with LTFS Bulk Transfer Standard, a Transfer Request XML document (3) is generated that specifies the desired transfer.	This functionality can be implemented as a downloadable software tool, or through a web-based interface provided by the cloud provider.

Transfer Step	Implementation Notes
3. The Source Transfer Manager (4) takes the Transfer Request XML (3), and creates a set of LTFS Volumes (5) containing the files specified in the XML. If the Transfer Request XML (3) is to be sent in-band, it is also stored as a file named "transfer.xml" in the root directory of each LTFS Volume (5).	This functionality is typically implemented as a downloadable software tool (which may be integrated with the Transfer Initiator). This is because web-based applications typically do not have access to local tape drives and libraries. When performing a transfer to a cloud, it is common to have the locally run software be integrated with the cloud provider's systems, as the destination environment is well-known. This permits the method by which the Transfer Request XML is sent, the transport of LTFS keys, and management of permissions to be pre-selected and automated. The cloud service provider may also generate shipping labels and manifests to facilitate tape asset management and tracking once the tapes have been sent to the provider. This allows the original tapes to be easily returned to the sender for re-use, or retained by the cloud provider for data protection or future retrieval by the sender.
4. The LTFS Volumes (5) are physically transported to the Destination Cloud.	This process is often mediated by the cloud service provider. For example, shipping labels for the tapes can be automatically generated from the cloud service provider's web interface, and information such as shipment tracking can be integrated and visible through the cloud service provider's web interface and via e-mail notifications to the Transfer Initiator. Once the tapes are received by the provider, they are loaded into one or more tape libraries. The presence of the tapes (identified by the barcode and Volume UUID specified in the Transfer Request XML) then triggers the next steps in the process.

Transfer Step	Implementation Notes
5. The Transfer Request XML (3) is received by the Destination Transfer Manager (6) in the Destination Cloud, either by reading from the LTFS Volumes (5) or out-of-band.	Cloud service providers can provide a secure channel for transferring the Transfer Request XML out-of-band.
6. If LTFS Volume encryption is used, the LTFS Volume keys are transported to the Destination Transfer Manager (6) out-of-band.	Cloud service providers can mediate the process of key generation and management, as they already have an existing secure communication channel with the Transfer Initiator.
7. The Destination Transfer Manager (6) reads the Transfer Request XML (3), and uses it to verify and merge the files on the LTFS Volumes (5) into the Destination Cloud Namespace (7).	
8. If requested in the Transfer Request XML (3), the Destination Transfer Manager (6) creates a Transfer Report XML document (8) that can be sent back to the Transfer Initiator (2)	The Cloud Service Provider can also provide a web-based report that provides a more user-friendly view into the transfer results.
9. The Transfer Initiator (2) receives the Transfer Report XML (8), and can use the included information to signal success or failure to the user, and optionally use the provided information to create a second transfer to allow recovery from various error conditions. (See Section 4 - Recovery from Failures)	An indication of success and failure would also be provided through the Cloud Service Provider's web interface and/or e-mail notifications.

2.3 LTFS Transfer from Cloud Source

Use cases exist where large quantities of data must be transferred from a public or private cloud. The LTFS Bulk Transfer Standard enables the workflow shown in Figure 4 for receive from cloud transfer of data.

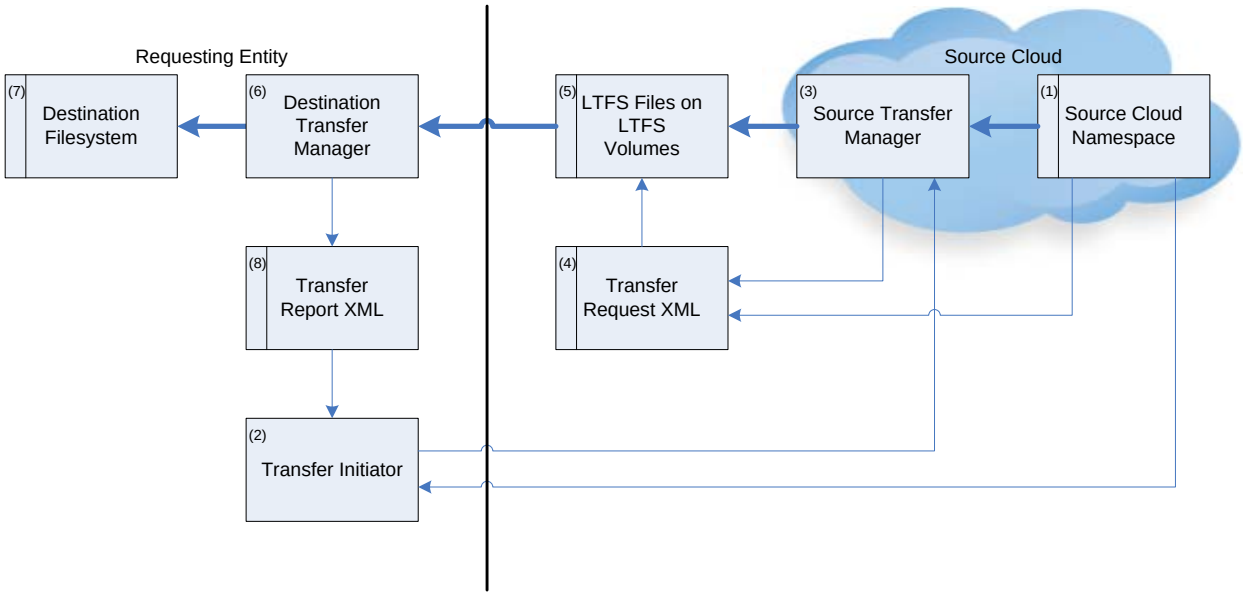

Figure 4 -LTFS Transfer from Cloud

Table 3 shows the steps performed as part of a LTFS Transfer from Cloud.

Table 3 - Steps in LTFS Transfer from Cloud

Transfer Step	Implementation Notes
1. A Transfer Initiator (2) desires to transfer files from a Source Cloud Namespace (1) to a local Destination Filesystem (7).	User signs in to their cloud account and can browse through their remotely stored files and objects.
2. The Transfer Initiator (2) selects the subset of files from the Source Cloud Namespace (1), and specifies how the files are to be merged into the local Destination Filesystem (7). This indication of the subset is sent to the Source Transfer Manager (3) resident in the Source Cloud.	User can select one or more containers/buckets, individual files/objects, or specify search criteria that matches against a set of files/objects. Typically the set of files/objects proposed to be retrieved via tape is indicated to the user, along with information such as total space required, estimated time to process, cost estimates, etc. Service providers may also provide options for job priority ("rush" vs. normal priority), delivery method, etc.

Transfer Step	Implementation Notes
3. The Source Transfer Manager (3) creates the Transfer Request XML document (4) to specify the desired transfer.	Once the user finalizes and commits the transfer, the cloud provider enumerates through the selected files/objects and builds a Transfer Request XML.
4. The Source Transfer Manager (3) creates a set of LTFS Volumes (5) containing the files specified in the Transfer Request XML (4). The Transfer Request XML (4) is stored as a file named "transfer.xml" in the root directory of each LTFS Volume (5).	The cloud provider selects available tapes in a tape library, formats the tapes with LTFS, and records the LTFS Volume UUIDs into the Transfer Request XML as files are copied onto the tapes.
5. The LTFS Volumes (5) are physically transported to recipient.	This process is managed by the cloud service provider. For example, shipping labels for the tapes are automatically generated, and information such as shipment tracking is integrated and visible through the cloud service provider's web interface and via e-mail notifications to the Transfer Initiator. Information such as the shipping address of the initiator is typically part of account information already known by the cloud provider.
6. If LTFS Volume encryption is used, the LTFS Volume keys are transported to the Destination Transfer Manager (6) out of band.	Cloud service providers can mediate the process of key storage and transfer through their existing client accounts and web interface.
7. The LTFS Volumes (5) are received by the Destination Transfer Manager (6), and the Transfer Request XML (4) is retrieved from the LTFS Volumes (5).	
8. The Destination Transfer Manager (6) reads the Transfer Request XML (4), and uses it to verify and merge the files on the LTFS Volumes (5) into the Destination Filesystem (7).	

Transfer Step	Implementation Notes
9. If requested in the Transfer Request XML (4), the Destination Transfer Manager (6) creates a Transfer Report XML document (8) that can be sent back to the Source Transfer Manager (3) in the Source Cloud.	The Transfer Report XML can be used by the Cloud Provider to send additional replacement tapes if tapes were missing, unreadable or corrupt.

2.4 LTFS Transfer between Clouds

Use cases exist where large quantities of data must be transferred between public and/or private clouds. The LTFS Bulk Transfer Standard enables the workflow shown in Figure 5 for cloud-to-cloud transfer of data.

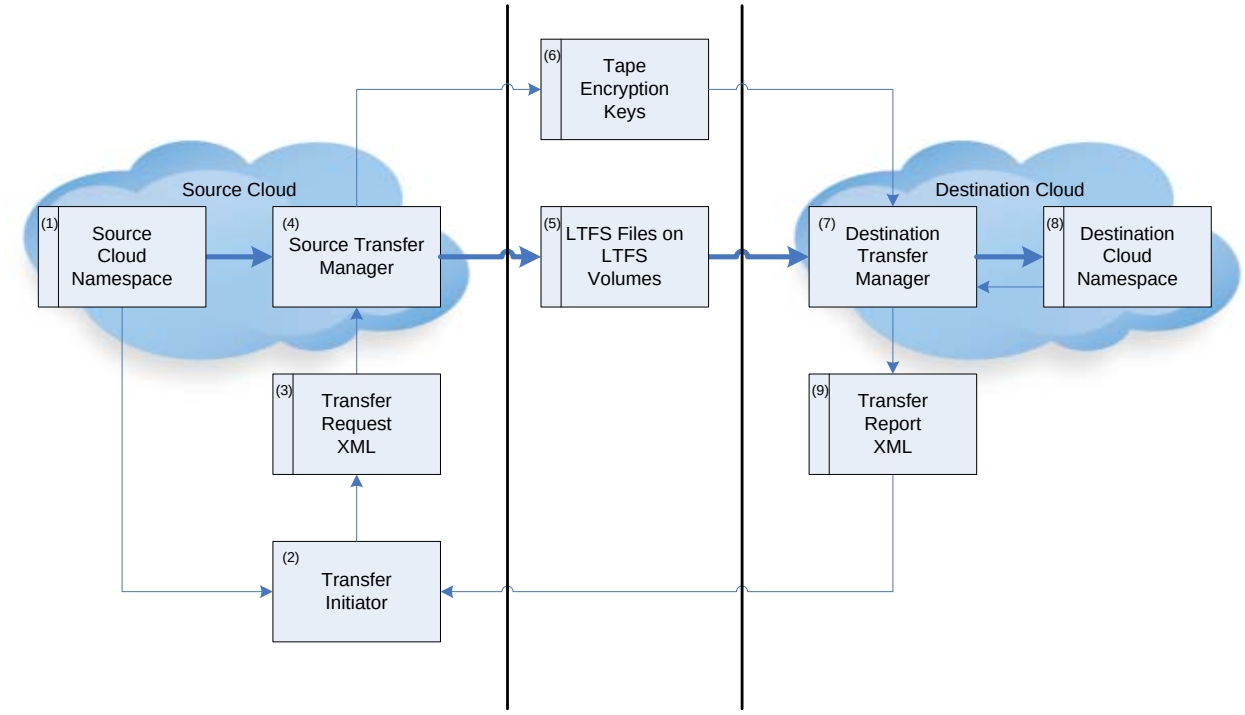

Figure 5 - Transfer between Clouds

Table 4 shows the steps performed as part of a cloud-to-cloud transfer.

Table 4 - Steps for Transfer between Clouds

Transfer Step	Implementation Notes
1. A Transfer Initiator (2) associated with the source cloud desires to transfer files and objects from local Source Cloud Namespace (1) to a remote Destination Cloud Namespace (8).	For cloud to cloud transfers, the need to transfer files and objects is typically policy driven. For example, a system could determine that it is more cost effective to transfer a large quantity of data between cloud sites via tape than to send it over a limited or congested WAN network link. In a second scenario, a transfer is automatically initiated when the permanent loss of a storage repository in a remote site is detected and violates the cloud's data protection policy.
2. The Transfer Initiator (2) selects the subset of files and objects from the Source Cloud Namespace (1), and specifies how the files and objects are to be merged into the remote Destination Cloud Namespace (8).	The selection of which files and objects to be transferred is typically performed automatically based on policy. For example, in the scenario where a loss of data has been automatically detected, the cloud would determine which files and objects were lost at the remote site, and use that information to select the files and objects are to be transferred.
3. The Transfer Initiator (2) creates a Transfer Request XML document (3) to specify the desired transfer.	
4. The Source Transfer Manager (4) creates a set of LTFS Volumes (5) containing the files and objects specified in the Transfer Request XML (3). The Transfer Request XML (3) is also stored as a file named "transfer.xml" in the root directory of each LTFS Volume (5).	
5. The LTFS Volumes (5) are physically transported to Destination Transfer Manager (7) in the Destination Cloud.	

Transfer Step	Implementation Notes
6. If Tape Volume encryption is used, the Tape Encryption Keys (6) are transported to the Destination Transfer Manager (7) out of band.	Tape encryption keys are typically managed using a secure key manager, such as a KMIP-based system. Assuming that the source and destination do not use the same key manager (or a federated key manager system), in order to securely and automatically transfer keys from the source cloud to the destination cloud, a key transfer facility must be provided. This is out of scope of this standard.
7. Using the provided Tape Encryption Keys (6), the Destination Transfer Manager (7) reads the Transfer Request XML (3) from the received LTFS Volumes (5).	
8. The Destination Transfer Manager (7) uses the Transfer Request XML (3), to verify and merge the files and objects contained on the LTFS Volumes (5) into the Destination Cloud Namespace (8).	
9. If requested in the Transfer Request XML (3), the Destination Transfer Manager (7) creates a Transfer Report XML document (9) that is sent back to the Source Cloud Transfer Initiator (2).	The mechanism by which the Transfer Report XML is returned to the Source Cloud is out of scope, but would typically be automated via an HTTP PUT API.
10. The Source Cloud Transfer Initiator (2) compares the Transfer Report XML (9) against the Transfer Request XML (3) to determine if any corrective actions are required.	The Transfer Report XML can be used by the Source Cloud Provider to send additional replacement tapes if tapes were missing, unreadable or corrupt.

2.5 Brokered LTFS Transfer between Clouds

Use cases exist where a cloud broker may mediate the transfer of large quantities of data between clouds. The LTFS Bulk Transfer Standard enables the workflow shown in Figure 6 for broker-mediated cloud-to-cloud transfers of data.

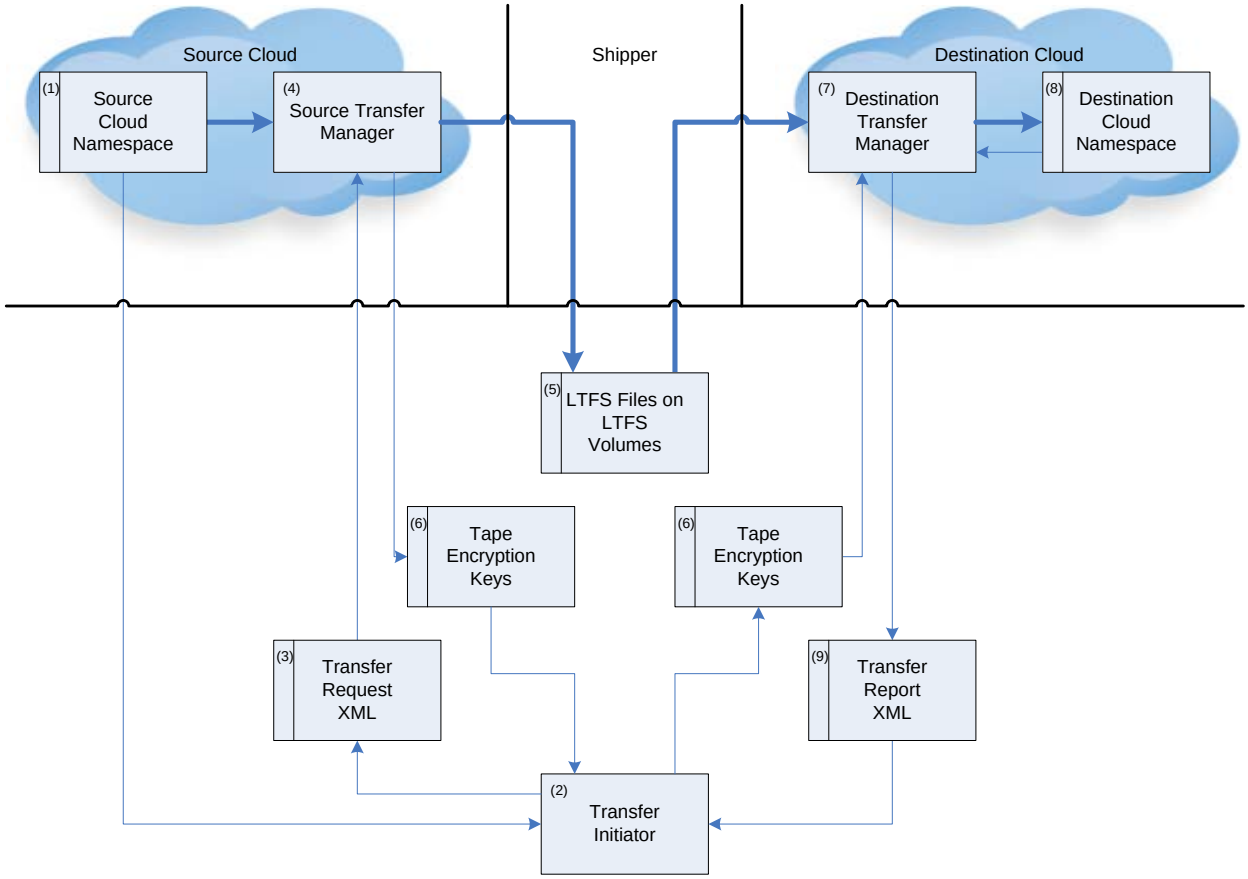

Figure 6 - Workflow for Broker-mediated Cloud-to-Cloud Transfer

The workflow is the same as described in section 2.4, except that the Transfer Initiator, key delivery and tape shipment are mediated by the cloud broker.

Table 5 - Steps for Broker-mediated Cloud-to-Cloud Transfer

Transfer Step	Implementation Notes
1. A Transfer Initiator (2) associated with the cloud broker desires to transfer files and objects from a Source Cloud Namespace (1) to a Destination Cloud Namespace (8).	
2. The Transfer Initiator (2) selects the subset of files and objects from the Source Cloud Namespace (1), and specifies how the files and objects are to be merged into the Destination Cloud Namespace (8).	The cloud broker typically uses RESTful APIs to inspect the namespaces of the source and destination clouds.
3. The Transfer Initiator (2) creates a Transfer Request XML document (3) to specify the desired transfer and submits the request to the Source Transfer Manager (4) in the Source Cloud.	The cloud broker typically uses RESTful APIs to submit a transfer request and to obtain status information about a pending transfer request.
4. The Source Transfer Manager (4) creates a set of LTFS Volumes (5) containing the files and objects specified in the Transfer Request XML (3). The Transfer Request XML (3) is also stored as a file named "transfer.xml" in the root directory of each LTFS Volume (5).	
5. The LTFS Volumes (5) are physically transported to Destination Transfer Manager (7) in the Destination Cloud, either directly or by way of the Cloud Broker.	
6. If Tape Volume encryption is used, the Tape Volume Keys (6) are transported to the Destination Transfer Manager (7) out of band.	The exchange of keying information may be mediated by the Cloud Broker.
7. Using the provided Tape Encryption keys, the Destination Transfer Manager (7) reads the Transfer Request XML (3) from the received LTFS Volumes (5).	The cloud broker typically uses RESTful APIs to send the Transfer Request XML and key information to the Destination Cloud.

Transfer Step	Implementation Notes
8. The Destination Transfer Manager (7) uses the Transfer Request XML (3), to verify and merge the files and objects contained on the LTFS Volumes (5) into the Destination Cloud Namespace (8).	
9. If requested in the Transfer Request XML (3), the Destination Transfer Manager (7) creates a Transfer Report XML document (9) that is sent back to the Source Cloud Transfer Initiator (2) via the Cloud Broker.	
10. The Source Cloud Transfer Initiator (2) compares the Transfer Report XML (9) against the Transfer Request XML (3) to determine if any corrective actions are required.	

3. Transfer Failures

In the process performing an LTFS Transfer, multiple error conditions may be encountered.

3.1 Insufficient Permission Errors

When an LTFS transfer specifies that a given file or directory is to be created, modified, or deleted in the destination namespace, the success of this operation is dependent on the permissions of the corresponding location within the destination namespace.

This failure can occur if the privileges of the user running the Destination Transfer Manager are insufficient to modify the destination namespace, or if the Destination Transfer Manager itself, when verifying permissions required to modify the destination namespace, determines that these permissions prohibit the operation.

3.2 Destination Write Errors

During the process of writing data to the destination namespace, a variety of errors may arise from the system providing the storage for the destination namespace. Examples of these errors include I/O errors, disrupted network connectivity, disk full/over-quota, etc.

3.3 Source Read Errors

During the process of reading data from the LTFS volumes, a variety of errors may arise from the LTFS storage system. Examples of these errors include being unable to mount a tape, I/O errors while reading the tape, tape inaccessibility due to contention, LTFS volume corruption, disrupted network connectivity, etc.

3.4 Item Missing Errors

During the process of transferring items from the LTFS volumes to the destination namespace, it is considered an error if an item specified to be transferred by the Transfer Request XML is not found on the indicated LTFS volume. This situation may occur due to modifications of the LTFS volumes subsequent to their creation or as a result of other errors that cause the file to no longer being visible.

3.5 Hash Mismatch Errors

During the process of transferring items from the LTFS volumes to the destination namespace, it is considered an error when the hash of a file transferred to the destination namespace does not match the hash of the file as specified in the Transfer Request XML. This situation may occur due to modifications of the LTFS volumes subsequent to their creation, or as a result of other errors resulting in the LTFS file's contents being altered or corrupted, or corruption during the process of writing the data to the destination namespace.

3.6 Volume Missing Errors

When the Destination Transfer Manager is initially enumerating the volumes associated with a transfer operation, the condition where not all of the volumes are accessible is considered an error. Examples of these errors include being unable to mount a tape, barcode mismatches, I/O errors while reading the tape, tape inaccessibility due to contention, LTFS volume corruption, disrupted network connectivity, etc.

The Destination Transfer Manager should first enumerate the set of tapes to ensure that all LTFS volumes described in the LTFS Transfer XML are present. This allows the Destination Transfer Manager to indicate when there are missing tapes, and allow the user to choose to proceed with the tape(s) considered missing, or wait until the missing tapes are found.

When tapes are considered missing, the Destination Transfer Manager shall be capable of creating a Transfer Report XML indicating which tapes are missing, and if the transfer was started or not.

3.7 Volume Decryption Errors

When a tape associated with an LTFS volume that is included as part of the LTFS Transfer is encrypted, the situation where the Destination Transfer Manager (or underlying tape management system) does not have access to the keys required to decrypt the tape contents to access the LTFS volume is considered an error.

The exchange of keying information required to access encrypted tapes is not specified in this standard.

4. Recovery from Failures

It is anticipated that LTFS Transfer Destination Manager shall operate in one of two error handling modes:

In "Preflight" mode, the Transfer Destination Manager shall verify the accessibility of all LTFS volumes, the existence of all files, and verify the integrity of each file before making any changes to the Destination Filesystem. This allows the detection of errors upfront, and reduces the probability of needing to perform a rollback.

In "Inline" mode, the Transfer Destination Manager shall verify the presence of tapes, files and file integrity as changes are being made to the Destination Filesystem. This mode of operation is more efficient, and is used when a rollback is not required when an error is encountered.

When a failure is encountered that cannot be resolved by the Destination Transfer Manager software, by the user running the Destination Transfer Manger software, or by the system administrator, the transfer shall fail, and if requested, details about the failure shall be included in the Transfer Report XML.

When the Transfer Initiator and/or Source Transfer Manager receives a Transfer Report XML that includes errors, the following recovery approaches can be used.

4.1 Insufficient Permission Errors

When a Transfer Report XML is returned that includes insufficient permission errors, the Transfer Initiator may suggest to the user performing the transfer that the ownership information associated with source content be squashed, which can be accomplished by specifying a "*" wildcard in the **sourceuserid** element in order to map all input user IDs to a single output user ID that has permissions to modify the destination namespace.

Retrying just the files that failed is accomplished by sending a second Transfer Request XML that refers to the files on existing tapes that failed.

4.2 Destination Write Errors

When a Transfer Report XML is returned that includes destination write errors, the Transfer Initiator can either retry the entire transfer, or retry transferring just the specific files that failed.

Retrying just the files that failed is accomplished by sending a second Transfer Request XML that refers to the files on existing tapes that failed.

If the entire transfer was aborted, it can be retried as-is.

4.3 Source Read Errors

When a Transfer Report XML is returned that includes source read errors, the Transfer Initiator can retry the transfer by creating new tapes to replace the tapes where the read errors occurred.

The new tapes and Transfer Request XML can then be sent to the Destination Transfer Manager, and when used together with the original tapes, allows the transfer operation to complete.

4.4 Item missing Errors

When a Transfer Report XML is returned that includes item missing errors, the Transfer Initiator can retry the transfer by creating additional tapes to add in the missing files, and generate a new Transfer Request XML referring to these added files.

The new tapes and Transfer Request XML can then be sent to the Destination Transfer Manager, and when used together with the original tapes, allows the transfer operation to complete.

4.5 Hash Mismatch Errors

When a Transfer Report XML is returned that includes hash mismatch errors, the Transfer Initiator can retry the transfer by either creating additional tapes to add in the corrupted files, and generate a new Transfer Request XML referring to these added files, or by creating new tapes to replace the tapes where the hash mismatch errors were encountered.

The new or additional tapes and new Transfer Request XML can then be sent to the Destination Transfer Manager, and when used together with the original tapes, allows the transfer operation to complete.

4.6 Volume Missing Errors

When a Transfer Report XML is returned that indicates that tapes were missing (determined when every file under a given volume is missing), the Transfer Initiator can re-create the missing tapes, and generate a new Transfer Request XML referring to the files on the new tapes. The new tapes can then be sent to the destination, where the transfer can be retried.

The new tapes and Transfer Request XML can then be sent to the Destination Transfer Manager, and when used together with the original tapes, allows the transfer operation to complete.

5. XML Definitions

5.1 Transfer Request XML

The Transfer Request is an XML data structure that describes information about the bulk transfer operation. The Transfer Request shall conform to the Transfer Request XML schema provided in Appendix A Transfer Request XML Schema. The Transfer Request shall be encoded using UTF-8 NFC.

5.1.1 Example Transfer Request XML

An example Transfer Request is shown in this section.

```
<?xml version="1.0" encoding="UTF-8"?>
<transferrequest version="1.0.0">
 <transferuuid>88481E70-DAF4-11E3-9C1A-0800200C9A66</transferuuid>
 <defaultoptions>
  <destinationpathprefix>/myContainer/</destinationpathprefix>
  <dirnotexistsaction>create</dirnotexistsaction>
  <direxistsaction>mergefromnewest</direxistsaction>
  <filenotexistsaction>create</filenotexistsaction>
  <fileexistsaction>replacewithnewest</fileexistsaction>
  <visibility>immediate</visibility>
  <erroraction>abort</erroraction>
 </defaultoptions>
 <usermapping>
   <user>
     <sourceuserid>jdoe</sourceuserid>
     <destuserid>john_doe</destuserid>
   </user>
 </usermapping>
 <transferreportcontents>all</transferreportcontents>
 <volumelist>
  <volume>
   <volumeuuid>d415fda2-5c9f-4d3d-9b97-43f3e8227ffe</volumeuuid>
   <mambarcode>AB1234L5</mambarcode>
   <directory>
    <name></name>
    <existsaction>preserve</existsaction>
    <contents>
     <file>
      <name>example.txt</name>
      <extendedattributes>
       <xattr>
        <x-attr-sha256sum>841ecd7a...cfe2c4af03c</x-attr-sha256sum>
       </xattr>
      </extendedattributes>
      <notexistsaction>create</notexistsaction>
      <existsaction>replacewithnewest</existsaction>
     </file>
     <file>
      <name>large_file.mp4</name>
      <extendedattributes>
       <xattr>
        <x-attr-sha256sum>c7f11288...82ac9f6e937</x-attr-sha256sum>
       </xattr>
      </extendedattributes>
      <existsaction>replacewithnewest</existsaction>
```

```
     </file>
    </contents>
   </directory>
  </volume>
  <volume>
   <volumeuuid>63753fab-f775-4d70-ae5e-6c416a867377</volumeuuid>
   <mambarcode>AB1235L5</mambarcode>
   <directory>
    <name></name>
    <contents>
     <file>
      <name>large_file.mp4</name>
      <extendedattributes>
       <xattr>
        <x-attr-sha256sum>798552d3...9458033b535b7b</x-attr-sha256sum>
       </xattr>
      </extendedattributes>
      <existsaction>replacewithnewest</existsaction>
     </file>
    </contents>
   </directory>
  </volume>
 </volumelist>
</transferrequest>
```

5.1.2 Transfer Request XML Elements

An **transferrequest** shall have exactly one **transferuuid**, one **defaultoptions**, may contain one **usermapping**, may contain one **transferreportcontents**, and shall have exactly one **volumelist**.

Each **transferuuid** shall contain a unique UUID generated corresponding to the specific transfer.

Each **defaultoptions** may contain the following elements:

> **destinationpathprefix**: this element indicates the location in the destination namespace where the directory or file shall be placed. The destination namespace shall be constructed by appending the LTFS directory hierarchy and file name to the destination path prefix.
>
> **dirnotexistsaction:** This element indicates the **notexistsaction** if a **notexistsaction** element is not specified for a given directory. If this element does not exist, the default **notexistsaction** for directories is "create".
>
> **direxistsaction:** This element indicates the **existsaction** if a **existsaction** element is not specified for a given directory. If this element does not exist, the default **notexistsaction** for directories is "mergefromnewest".
>
> **filenotexistsaction:** This element indicates the **notexistsaction** if a **notexistsaction** element is not specified for a given file. If this element does not exist, the default **notexistsaction** for directories is "create".

file**existsaction:** This element indicates the **existsaction** if a **existsaction** element is not specified for a given file. If this element does not exist, the default **notexistsaction** for directories is "replaceifnewest".

visibility: This element indicates if the changes to the destination namespace become visible immediately or only when the transfer request has been fully processed. The following values are defined:

"immediate"	Changes to the destination namespace shall become visible while the transfer is still in progress.
"completion"	Changes to the destination namespace shall become visible only once the entire transfer operation has completed.

erroraction: This element indicates what actions shall be taken when an error is encountered. The following values are defined:

"abort"	The processing of the transfer request shall be halted when an error is encountered, with the destination namespace being left as-is.
"rollback"	The processing of the transfer request shall be halted when an error is encountered, with the destination namespace being restored to its state at the beginning of the transfer.
"continue"	The processing of the transfer request shall continue when an error is encountered.

Each **usermapping** may contain one or more **user** elements:

Each **user** shall contain one **sourceuserid** and one **destuserid**.

sourceuserid:	A UTF8 string that indicates the user identifier on the LTFS transfer volume that is to be mapped to the corresponding **destuserid**. The value "*" matches against all source user ID.
destuserid:	A UTF8 string that indicates the user identifier on the destination namespace.

Each **transferreportcontents** indicates what information is returned in a transfer report, andshall contain one of the following values:

"all"	A transfer report shall be generated and shall contain information for every file and directory specified in the transfer request.

"changes"	A transfer report shall be generated and shall contain information for every file and directory created, updated or merged into the destination namespace.
"error"	A transfer report shall be generated and shall contain information for every file and directory where an error was encountered when creating, updating or merging into the destination namespace.
"none"	No transfer report shall be generated.

A **volumelist** shall have one or more **volume** element.

Each **volume** element shall contain all of the following elements:

volumeuuid: this element shall contain a universally unique identifier (UUID) value that uniquely identifies the LTFS Volume to which the Index is written. The value of the **volumeuuid** element must conform to the format definition shown in Section 5.8 UUID format in the LTFS standard. The **volumeuuid** value shall match the value of the **volumeuuid** element in the LTFS Labels written to the LTFS Volume.

mambarcode: The MAM attribute value stored as BARCODE which must correspond to the physical label on the cartridge.

directory: this element corresponds to the “root” directory element in the Index. The content of this element is described later in this section.

Each **directory** element shall have exactly one **name**, may contain one **extendedattributes**, may contain one **destinationpathprefix**, may contain one **notexistsaction**, may contain one **existsaction**, may contain one **visibility**, may contain one **erroraction**, and contains exactly one **contents** element.

Each **contents** element shall have zero or more **directory** elements and shall have zero or more **file** elements.

Each **file** element shall have exactly one **name**, may contain one **extendedattributes**, may contain one **destinationpathprefix**, may contain one **notexistsaction**, may contain one **existsaction**, may contain one **visibility**, and may contain one **erroraction**.

Each **name** element shall contain an LTFS name as defined in section 5.4 Name Format of the LTFS standard.

Each **extendedattributes** element shall contain exactly one **xattr** element. Only the hash extended attributes shall be included. When the hash extended attribute is present and the hash of the file on the LTFS transfer does not match the specified hash, this condition shall be treated as an error.

Each **destinationpathprefix** shall contain a UTF-8 string corresponding to a path on the transfer destination.

Each **notexistsaction** shall contain a value that indicates the behaviour if there is not an equivalent item in the destination namespace. If this element is omitted, then use the default behaviour as specified in the defaultoptions element.The following values are defined:

"create"	Create the item in the destination namespace.
"skip"	Leave the destination file or directory as-is, and if a directory, leave all contents as-is.

Each **existsaction** shall contain a value that indicates how conflicts are resolved when an identically named file or directory exists in the same location in both the LTFS transfer volume and in the destination namespace. If this element is omitted, then use the default behaviour as specified in the defaultoptions element. The following values are defined:

"preserve"	Leave the destination file or directory as-is, and if a directory, evaluate contents based on the corresponding XML elements.
"skip"	Leave the destination file or directory as-is, and if a directory, leave all contents as-is.
"replacealways"	Properties, attributes and children of the directory in the destination namespace shall be replaced by the properties, attributes and children of the LTFS directory. The file in the destination namespace shall be completely replaced by the LTFS file.
"replacewithnewest"	Properties, attributes and children of the directory in the destination namespace shall be replaced by the properties, attributes and children of the LTFS directory only if the source directory has a newer timestamp than the destination directory. The file in the destination namespace shall be completely replaced by the LTFS file only if the source file has a newer timestamp than the destination file.
"replacewitholdest"	Properties, attributes and children of the directory in the destination namespace shall be replaced by the properties, attributes and children of the LTFS directory only if the source directory has an equal or older timestamp than the destination directory. The file in the destination namespace shall be completely replaced by the LTFS file only if the source file has an equal or older timestamp than the destination file.

"mergealways"	Properties and attributes of the LTFS file or directory shall be merged into the properties and attributes of the file or directory in the destination namespace.
"mergefromnewest"	Properties and attributes of the LTFS file or directory shall be merged into the properties and attributes of the file or directory in the destination namespace only if the LTFS file or directory has a newer timestamp than the destination namespace file or directory.
"mergefromoldest"	Properties and attributes of the LTFS file or directory shall be merged into the properties and attributes of the file or directory in the destination namespace only if the LTFS file or directory has an equal or older timestamp than the destination namespace file or directory.

5.2 Transfer Report XML

The Transfer Report is an XML data structure that describes the result of a bulk transfer operation. The Transfer Report shall conform to the Transfer Report XML schema provided in Appendix B Transfer Report XML Schema. The Transfer Report shall be encoded using UTF-8 NFC.

5.2.1 Example Transfer Report XML

An example Transfer Report is shown in this section.

```
<?xml version="1.0" encoding="UTF-8"?>
<transferresponse version="1.0.0">
 <transferuuid>88481E70-DAF4-11E3-9C1A-0800200C9A66</transferuuid>
 <transferreportcontents>all</transferreportcontents>
 <volumelist>
  <volume>
   <volumeuuid>d415fda2-5c9f-4d3d-9b97-43f3e8227ffe</volumeuuid>
   <mambarcode>AB1234L5</mambarcode>
   <directory>
    <name></name>
    <transferresult>success</transferresult>
    <transferoperation>preserve</transferoperation>
    <contents>
     <file>
      <name>example.txt</name>
      <transferresult>success</transferresult>
      <transferoperation>create</transferoperation>
     </file>
     <file>
      <name>large_file.mp4</name>
      <transferresult>success</transferresult>
      <transferoperation>replace</transferoperation>
     </file>
    </contents>
   </directory>
  </volume>
  <volume>
   <volumeuuid>63753fab-f775-4d70-ae5e-6c416a867377</volumeuuid>
   <mambarcode>AB1235L5</mambarcode>
```

```
   <directory>
    <name></name>
    <transferresult>success</transferresult>
    <transferoperation>preserve</transferoperation>
    <contents>
     <file>
      <name>large_file.mp4</name>
      <transferresult>success</transferresult>
      <transferoperation>replace</transferoperation>
     </file>
    </contents>
   </directory>
  </volume>
 </volumelist>
</transferresponse>
```

5.2.2 Transfer Request XML Elements

An **transferresponse** shall have exactly one **transferuuid**, one **transferreportcontents** and one **volumelist**.

Each **transferuuid** shall contain a unique UUID corresponding to the transfer being rep.

transferreportcontents: This element indicates what information was requested to be included in the transfer report. The value shall match the value in the **transferreportcontents** element in the corresponding transfer request XML.

A **volumelist** shall have one or more **volume** element.

Each **volume** element shall contain all of the following elements:

volumeuuid: As defined in the Transfer Request XML.

mambarcode: As defined in the Transfer Request XML.

directory: this element corresponds to the “root” directory element in the Index. The content of this element is described later in this section.

Each **directory** element shall have exactly one **name**, one **transferresults**, one **transferoperation**, and one **contents** element.

transferresult: this element indicates the result of the transfer operation. The following values are defined:

"success":	Indicates that the operation completed successfully.
"insufficient permissions":	Insufficient permissions to create, update or replace the corresponding file or directory on the destination namespace.

"destination write error":	Write error associated with the destination namespace.
"source read error":	Read error associated with the LTFS volume.
"item missing":	Item specified in Transfer Request XML missing from LTFS volume.
"hash mismatch":	Hash of file on LTFS volume does not match file hash in XML transfer request XML.
"volume missing":	Volume specified in Transfer Request XML not available.
"volume decryption error":	Volume specified in Transfer Request XML could not be decrypted.
"other error":	An error not defined in this standard was encountered.

transferoperation: this element indicates the transfer operation that was performed or was attempted to be performed. The following values are defined:

"preserve":	The file or directory in the destination namespace was to be left as-is.
"replace":	The file or directory in the destination namespace was to be replaced with the LTFS file or directory
"merge":	The file or directory in the destination namespace was to be merged with the LTFS file or directory
"create":	The file or directory in the destination namespace was to be created using the corresponding LTFS file or directory

Each **contents** element shall have zero or more **directory** elements and shall have zero or more **file** elements.

Each **file** element shall have exactly one **name**, one **transferresults**, and one **transferoperation**.

6. Security Considerations

6.1 Identify Verification

Verification of the identity and source of a transfer is out of scope of this standard. It is up to the receiver of a LTFS transfer to verify that the tapes and Transfer Request XML are associated with a valid account and user.

6.2 Key management

Key management is out of the scope of this standard. It is up to the sender and receiver to coordinate to securely exchange keys in order to allow the LTFS volumes to be read when performing a transfer.

6.3 Weaponized Transfers

6.3.1 Deletion

A transfer request, if permitted, can overwrite existing files and directories. This can be used to erase information.

For example, the following Transfer Request XML will erase all files in the destination file namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<transferrequest version="1.0.0">
 <transferuuid>A28B4B40-DAF4-11E3-9C1A-0800200C9A66</transferuuid>
 <defaultoptions/>
 <volumelist>
  <volume>
   <volumeuuid>d415fda2-5c9f-4d3d-9b97-43f3e8227ffe</volumeuuid>
   <mambarcode>AB1234L5</mambarcode>
   <directory>
    <name></name>
    <existsaction>replacealways</existsaction>
    <contents>
    </contents>
   </directory>
  </volume>
 </volumelist>
</transferrequest>
```

Software that implements and performs an LTFS transfer to the destination namespace should be run with minimal privileges, should honor the ACLs and associated permissions of existing files and directories, and should be run within the namespace scope of a given client.

6.3.2 Sparse File Expansion

If sparse files are not supported on the Transfer Destination, restoring a sparse file can be used to exhaust all available space on the Transfer Destination Namespace.

6.3.3 Compression Expansion

Large files that are highly compressible (such as TBs of zeros) can be used to exhaust all available space on the Transfer Destination Namespace.

6.3.4 Block Reference Expansion

An LTFS index partition can define multiple files that all point to common file data. Restoring these files can exhaust all available space on the Transfer Destination Namespace when expanded.

6.4 Recommendations

1. A cloud provider should allow transfers where the destination is a new namespace, or a non-live "copy" of the namespace. This allows the transfer to be executed, then the resulting namespace inspected before the live namespace is replaced by the new or updated namespace.

2. Preflight should check for sparse files, compression file expansion and block reference expansion before reading file data as part of processing the transfer operation.

3. Preflight should verify space requirements on the destination before processing the transfer operation.

Appendix A. Transfer Request XML Schema

The Transfer Request XML is defined according to the below XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:annotation>
 <xsd:documentation xml:lang="en">
  XML Schema for a LTFS Transfer Request
 </xsd:documentation>
</xsd:annotation>

<xsd:element name="transferrequest" type="transferrequesttype"/>

<xsd:complexType name="transferrequesttype">
 <xsd:sequence>
  <xsd:element name="transferuuid" type="uuidtype"/>
  <xsd:element name="defaultoptions" type="defaultoptionstype"/>
  <xsd:element name="usermapping" type="usermappingtype" minOccurs="0"/>
  <xsd:element name="transferreportcontents" type="transferreportcontentstype"
minOccurs="0"/>
  <xsd:element name="volumelist" type="volumelisttype"/>
 </xsd:sequence>
 <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="transferreportcontentstype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="all"/>
  <xsd:enumeration value="changes"/>
  <xsd:enumeration value="error"/>
  <xsd:enumeration value="none"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="defaultoptionstype">
 <xsd:sequence>
  <xsd:element name="destinationpathprefix" type="xsd:string" minOccurs="0"/>
  <xsd:element name="dirnotexistsaction" type="notexistsactiontype" minOccurs="0"/>
  <xsd:element name="direxistsaction" type="existsactiontype" minOccurs="0"/>
  <xsd:element name="filenotexistsaction" type="notexistsactiontype" minOccurs="0"/>
  <xsd:element name="fileexistsaction" type="existsactiontype" minOccurs="0"/>
  <xsd:element name="visibility" type="visibilitytype" minOccurs="0"/>
  <xsd:element name="erroraction" type="erroractiontype" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="notexistsactiontype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="create"/>
  <xsd:enumeration value="skip"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="existsactiontype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="preserve"/>
  <xsd:enumeration value="skip"/>
  <xsd:enumeration value="replacealways"/>
```

```
  <xsd:enumeration value="replacewithnewest"/>
  <xsd:enumeration value="replacewitholdest"/>
  <xsd:enumeration value="mergealways"/>
  <xsd:enumeration value="mergefromnewest"/>
  <xsd:enumeration value="mergefromoldest"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="visibilitytype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="immediate"/>
  <xsd:enumeration value="completion"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="erroractiontype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="abort"/>
  <xsd:enumeration value="rollback"/>
  <xsd:enumeration value="continue"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="usermappingtype">
 <xsd:sequence>
  <xsd:element name="user" type="usertype" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="usertype">
 <xsd:sequence>
  <xsd:element name="sourceuserid" type="xsd:string"/>
  <xsd:element name="destuserid" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="volumelisttype">
 <xsd:sequence>
  <xsd:element name="volume" type="volumetype" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="volumetype">
 <xsd:sequence>
  <xsd:element name="volumeuuid" type="uuidtype"/>
  <xsd:element name="mambarcode" type="xsd:string"/>
  <xsd:element name="directory" type="directorytype"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="uuidtype">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{12}"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="directorytype">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="extendedattributes" type="extendedattributestype" minOccurs="0"/>
  <xsd:element name="destinationpathprefix" type="xsd:string" minOccurs="0"/>
  <xsd:element name="notexistsaction" type="notexistsactiontype" minOccurs="0"/>
```

```
  <xsd:element name="existsaction" type="existsactiontype" minOccurs="0"/>
  <xsd:element name="visibility" type="visibilitytype" minOccurs="0"/>
  <xsd:element name="erroraction" type="erroractiontype" minOccurs="0"/>
  <xsd:element name="contents" type="contentstype"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="contentstype">
 <xsd:sequence>
  <xsd:element name="directory" type="directorytype" minOccurs="0"
maxOccurs="unbounded"/>
  <xsd:element name="file" type="filetype" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="filetype">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="extendedattributes" type="extendedattributestype" minOccurs="0"/>
  <xsd:element name="destinationpathprefix" type="xsd:string" minOccurs="0"/>
  <xsd:element name="notexistsaction" type="notexistsactiontype" minOccurs="0"/>
  <xsd:element name="existsaction" type="existsactiontype" minOccurs="0"/>
  <xsd:element name="visibility" type="visibilitytype" minOccurs="0"/>
  <xsd:element name="erroraction" type="erroractiontype" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="extendedattributestype">
 <xsd:sequence>
  <xsd:element name="xattr" type="xattrtype" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="xattrtype">
 <xsd:sequence>
  <xsd:element name="x-attr-sha256sum" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

Appendix B. Transfer Report XML Schema

The Transfer Report XML is defined according to the below XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:annotation>
 <xsd:documentation xml:lang="en">
  XML Schema for a LTFS Transfer Report
 </xsd:documentation>
</xsd:annotation>

<xsd:element name="transferresponse" type="transferresponsetype"/>

<xsd:complexType name="transferresponsetype">
 <xsd:sequence>
  <xsd:element name="transferuuid" type="uuidtype"/>
  <xsd:element name="transferreportcontents" type="transferreportcontentstype"/>
  <xsd:element name="volumelist" type="volumelisttype"/>
 </xsd:sequence>
 <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="transferreportcontentstype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="all"/>
  <xsd:enumeration value="changes"/>
  <xsd:enumeration value="error"/>
  <xsd:enumeration value="none"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="volumelisttype">
 <xsd:sequence>
  <xsd:element name="volume" type="volumetype" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="volumetype">
 <xsd:sequence>
  <xsd:element name="volumeuuid" type="uuidtype"/>
  <xsd:element name="mambarcode" type="xsd:string"/>
  <xsd:element name="directory" type="directorytype"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="uuidtype">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{12}"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="directorytype">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="transferresult" type="transferresulttype"/>
  <xsd:element name="transferoperation" type="transferoperationtype"/>
  <xsd:element name="contents" type="contentstype"/>
```

```
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="transferresulttype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="success"/>
  <xsd:enumeration value="insufficient permissions"/>
  <xsd:enumeration value="destination write error"/>
  <xsd:enumeration value="source read error"/>
  <xsd:enumeration value="item missing"/>
  <xsd:enumeration value="hash mismatch"/>
  <xsd:enumeration value="volume missing"/>
  <xsd:enumeration value="volume decryption error"/>
  <xsd:enumeration value="other error"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="transferoperationtype">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="preserve"/>
  <xsd:enumeration value="replace"/>
  <xsd:enumeration value="merge"/>
  <xsd:enumeration value="create"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="contentstype">
 <xsd:sequence>
  <xsd:element name="directory" type="directorytype" minOccurs="0"
maxOccurs="unbounded"/>
  <xsd:element name="file" type="filetype" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="filetype">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="transferresult" type="transferresulttype"/>
  <xsd:element name="transferoperation" type="transferoperationtype"/>
 </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```