ASC Media Hash List (ASC MHL)

Advanced Data Management Subcommittee

THE AMERICAN SOCIETY OF CINEMATOGRAPHERS

Specification v1.0

15. March, 2022

Table of Contents

Introduction

It is critical to create secure and reliable backups of on-set media content to ensure lossless transfer during production and post processes. However, verifying the integrity of these files as they move between various facilities is currently a challenging, laborious, and error-prone task. There is no mechanism to track when and where a file was damaged or how many times a file has been duplicated. These challenges stem from use of different tools and processes between sending and receiving facilities. The American Society of Cinematographers (ASC) has developed the ASC Media Hash List (ASC MHL) specification to standardize the media transfer process with the goal of injecting consistency and efficiency in media workflows.

After media is recorded in camera, it needs to be copied to other storage devices, so that camera cards can be cleared and reused. Camera cards are often organized by shooting day as part of being offloaded to on-set storage. Then the media content needs to travel to several facilities in order to affect post-production processes on them. Depending on the receiving facility's ingest policies, checksums may or may not be created for the received content files. Even if checksums are created, their location on the transfer device and the underlying method to generate them could be inconsistent between sending and receiving entities. ASC has designed the ASC MHL specification to solve these underlying problems while giving facilities freedom to develop media handling policies that fit their unique needs.

The ASC MHL specification accommodates various common hash algorithms, allowing productions and post facilities to choose the algorithm that suits their unique workflows. ASC MHL mandates the presence of critical information, defines how and where this information is stored, and records the hashes for the media in a human and machine-readable ASC MHL Manifest file. By requiring that a new ASC MHL Manifest be created for every copy, the specification creates a chain of custody, thereby allowing for accurate tracking of damaged and duplicated files.

1. Scope

This document specifies format definitions and operations of ASC Media Hash List (ASC MHL), for the exchange and processing of media hashes and associated metadata. It contains the information required to implement an "ASC MHL-compliant" software system.

ASC MHL is intended for use in media production workflows and has been optimized to support the tracking of interactions and changes within a data set over the course of multi-step data transfer. A common example is movement of camera original media from a production set to post production teams and multiple backup targets with end-to-end verification.

While ASC MHL may be applicable in file-based workflows beyond media, such applications are outside the scope of this document. For example ASC-MHL may not be ideal for indexing of filesystems for search optimization or maintaining chain of custody in N-way sharing of dynamic research data sets.

Note: This specification is called “ASC MHL” to unambiguously distinguish it from a predecessor specification that predates ASC MHL and has been specified outside of the ASC as “MHL”.

2. Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:”

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document. The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

The following font/font color formatting is used throughout this document:

- Consolas font is used for file or directory names, e.g. `ascmhl_chain.xml`
- Purple Consolas font is used for XML elements, attributes and values, e.g. `HashType`

3. References

World Wide Web Consortium (W3C) (26 November 2008). Extensible Markup Language (XML) 1.0 (Fifth Edition)

World Wide Web Consortium (W3C) (28 October 2004). XML Schema Part 1: Structures (Second Edition)

World Wide Web Consortium (W3C) (28 October 2004). XML Schema Part 2: Datatypes (Second Edition)

SMPTE ST 2114:2017, Unique Digital Media Identifier (C4 ID)

xxHash https://github.com/Cyan4973/xxHash

Internet Engineering Task Force (IETF) (September 2001). RFC3174 - US Secure Hash Algorithm 1 (SHA1)

Internet Engineering Task Force (IETF) (April 1992). RFC1321 - The MD5 Message-Digest Algorithm

Internet Engineering Task Force (IETF) (October 2006). RFC4648 - The Base16, Base32, and Base64 Data Encodings

4. Glossary

4.1. Scope of an ASC MHL History/Manifest: the directory tree covered by the ASC MHL History/Manifest.

4.2. Managed Data Set: The collection of files/folders within the scope of an ASC MHL History with one common root directory.

4.3. Data Management Process: Any process that adds new items or removes or changes existing items within a managed data set, e.g. copying, moving or deleting files, or updating embedded metadata structures.

4.4. Hash Record: A collection of information about a file or directory, consisting of the path to the file or directory, related information such as file size, and the associated, qualified hash values.

4.5. ASC MHL tool: A software system or tool implementing the required formats and behaviors defined by this specification.

4.6. MD5: The algorithm specified in [RFC1321].

4.7. SHA1: The algorithm specified in [RFC3174].

4.8. C4: The algorithm specified in [SMPTE ST 2114].

4.9. XXH64: The algorithm specified as XXH64 in [xxHash].

4.10. XXH3: The algorithm specified as XXH3 in [xxHash].

4.11. XXH128: The algorithm specified as XXH128 in [xxHash].

5. Concepts and Semantics

5.1. General

ASC MHL allows for the creation of hash records for data sets and the subsequent verification of such data sets based on established hashing algorithms. ASC MHL also supports the initiation and maintenance of a chain of custody by tracking each copy made after the creation of a managed data set. The application of ASC MHL is to document hash records in ASC MHL Manifest files and, optionally, keep track of the history of a managed data set in an ASC MHL History.

5.2. ASC MHL Manifest

An ASC MHL Manifest is a file, as detailed in ASC MHL Manifest File, that contains hash records for one or more files and/or directories within its scope. The scope of an ASC MHL Manifest is defined as the directory containing the files and directories of the managed data set of the ASC MHL Manifest. File system paths relative to the scope are used for hash records in the ASC MHL Manifest. An ASC MHL Manifest shall not contain hashes for files or directories outside of its scope.

An ASC MHL Manifest is part of an ASC MHL History when it is located in a directory named `ascmhl` alongside an ASC MHL Chain file. The `ascmhl` directory must reside in the root level of the ASC MHL History's scope.

If an ASC MHL Manifest is not part of an ASC MHL History, it is considered stand-alone and therefore, its scope cannot be derived by its location in the file system. The scope is also not explicitly specified in the ASC MHL Manifest and should be user-provided whenever needed.

Example: A data set is shipped on an USB drive while the associated ASC MHL Manifest is sent via email. In order to verify the data set based on the ASC MHL Manifest, an user will need to point an ASC MHL tool to the location of the data set in the file system and make the connection to the associated ASC MHL Manifest.

The hashes recorded in a given ASC MHL Manifest represent a snapshot of the corresponding managed data set at a given moment in time. Once created, ASC MHL Manifest files are immutable and shall not be altered.

An ASC MHL Manifest may contain more than one hash value for a given file or directory, each computed with a hash algorithm supported by ASC MHL (see Appendix D for details). Every hash is labeled as either `original`, `verified`, or `failed` using the `action` attribute of `HashFormatType` (see HashFormatType).

- `original` indicates that the hash is the initial hash for a file or directory within the ASC MHL History

- `verified` indicates that the hash was computed directly from the current copy of the file and verified against the file's hash recorded in a previous ASC MHL Manifest generation
- `failed` indicates that the hash computed from the current copy of the file does not match the record in a previous ASC MHL Manifest generation and that the file is therefore not identical to its previous copy

In addition to the list of hashes, an ASC MHL Manifest contains sections with information about the creator and the process used to create the ASC MHL Manifest. It may optionally contain references to other ASC MHL Manifests (see Nested ASC MHL Histories), as well as user defined metadata. See HashListType for details.

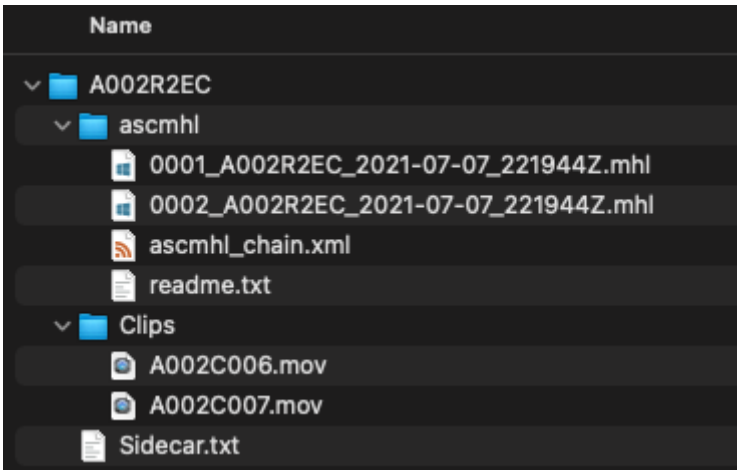
5.3. ASC MHL History

5.3.1. General

An ASC MHL History is defined as an ASC MHL Chain and all ASC MHL Manifests referenced from that ASC MHL Chain, stored in a directory named `ascmhl`.

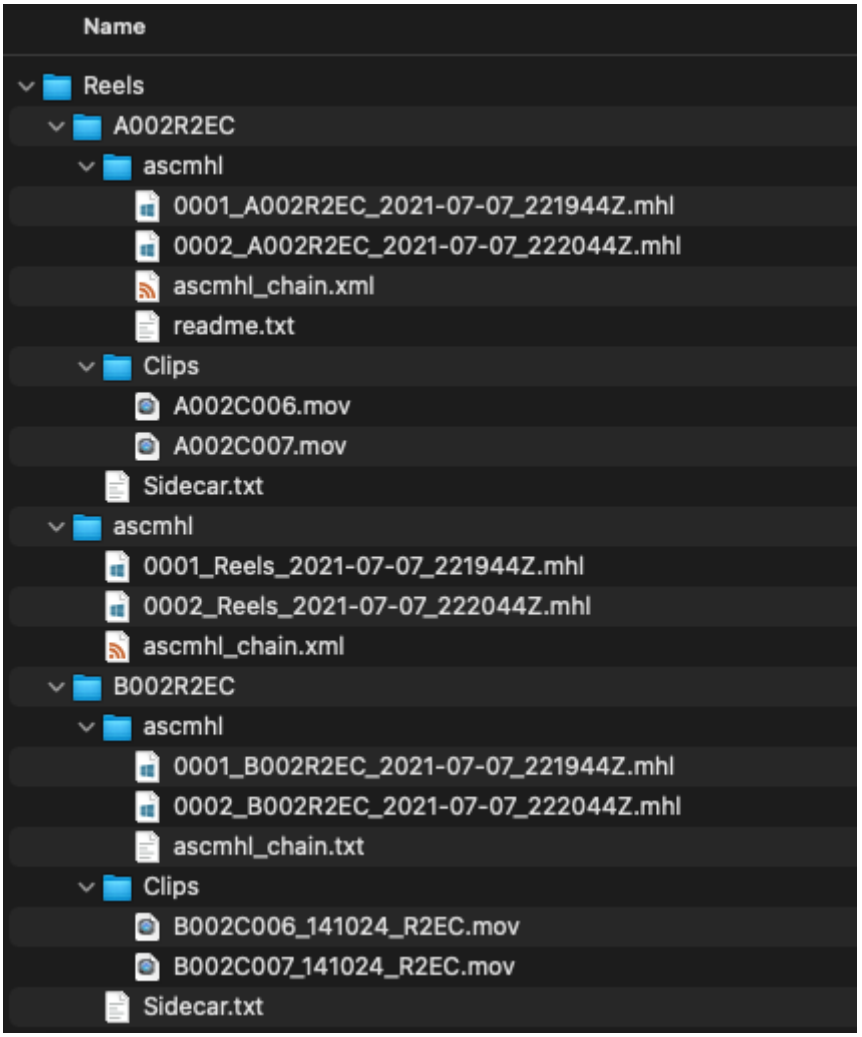
The `ascmhl` directory is created at the top level of a media directory, so that the ASC MHL History is automatically transferred whenever the media directory is copied or moved. That media directory is referred to as the scope of the ASC MHL History and the files/subdirectories inside the media directory are referred to as the managed data set. All ASC MHL Manifests within a given ASC MHL History shall cover the same scope, i.e. hash record paths across the ASC MHL Manifests shall be relative to the same location in the file system.

The `ascmhl` directory can include one optional text file, called `README.txt`. The file shall be UTF-8 encoded. The `README.txt` file can include any human-readable text information, for example general documentation about verification, links to ASC MHL tools, or other references. The `README.txt` file is not part of the history and shall not include any information already stored in the ASC MHL History (i.e in the ASC MHL Manifest and ASC MHL Chain files).



Example directory structure with media files, `ascmhl` directory, ASC MHL Manifests and ASC MHL Chain.

5.3.2. Nested ASC MHL Histories



Example folder structure of a nested ASC MHL History

ASC MHL Histories can be nested, i.e. ASC MHL Histories may be present within the scope of a given ASC MHL History. Nested ASC MHL Histories should be referenced in ASC MHL Manifests (see ReferencesType and HashListReferenceType) to establish a relationship between higher-level ASC MHL Histories and nested ASC MHL Histories.

The hash record of a file or directory shall be recorded in the ASC MHL History closest to it in terms of the file system hierarchy. This ensures that the entire history of a managed file/directory is documented in a single (i.e., the closest) ASC MHL History. Consequently, higher-level ASC MHL Histories only store hashes of files/directories outside of any of the subdirectories that contain nested ASC MHL Histories. But the references to nested ASC MHL Histories establish a connection between higher-level ASC MHL Histories and the files/directories covered by these nested ASC MHL Histories.

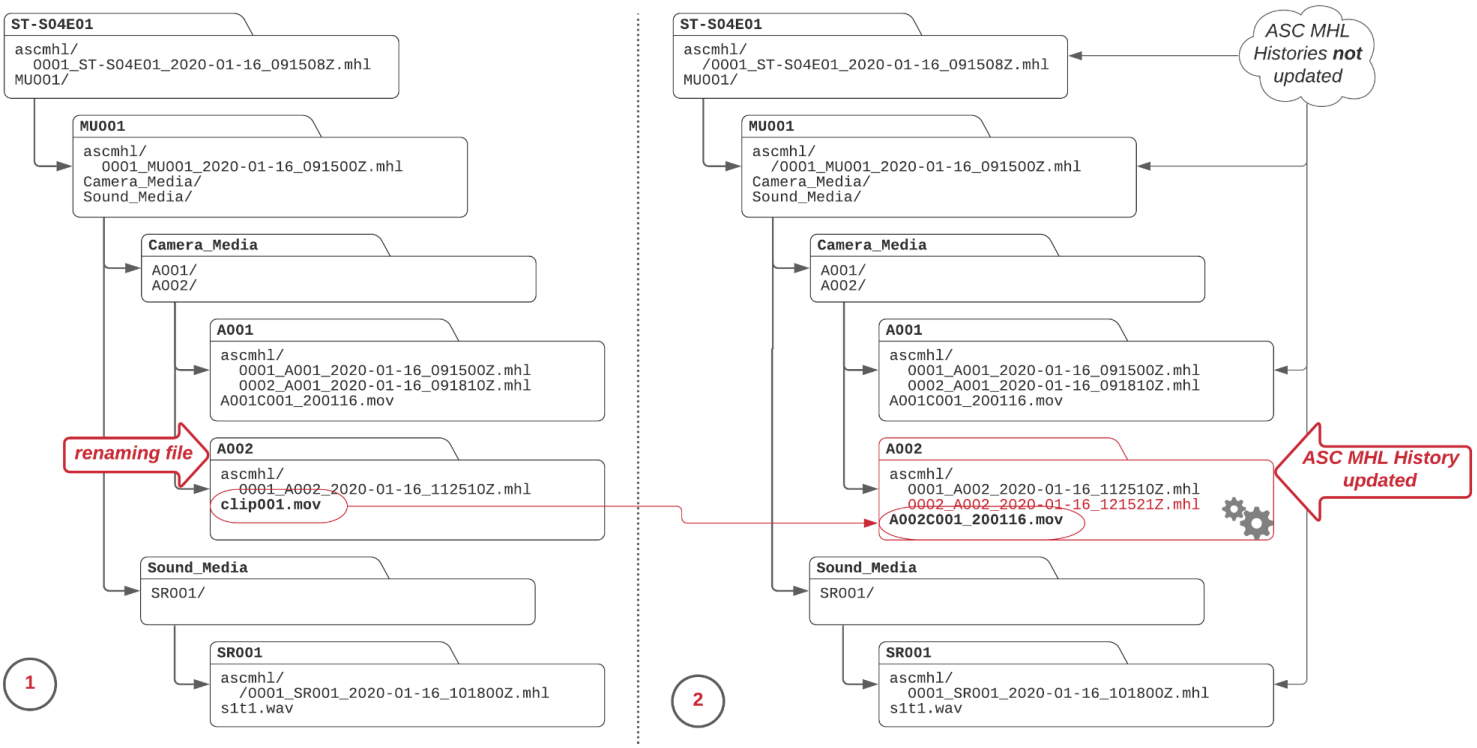
All information for a managed data set is documented between its ASC MHL History and nested ASC MHL Histories contained within its scope. When ASC MHL Histories contain at least one

nested ASC MHL History, generational changes are applied to nested ASC MHL Histories as follows:

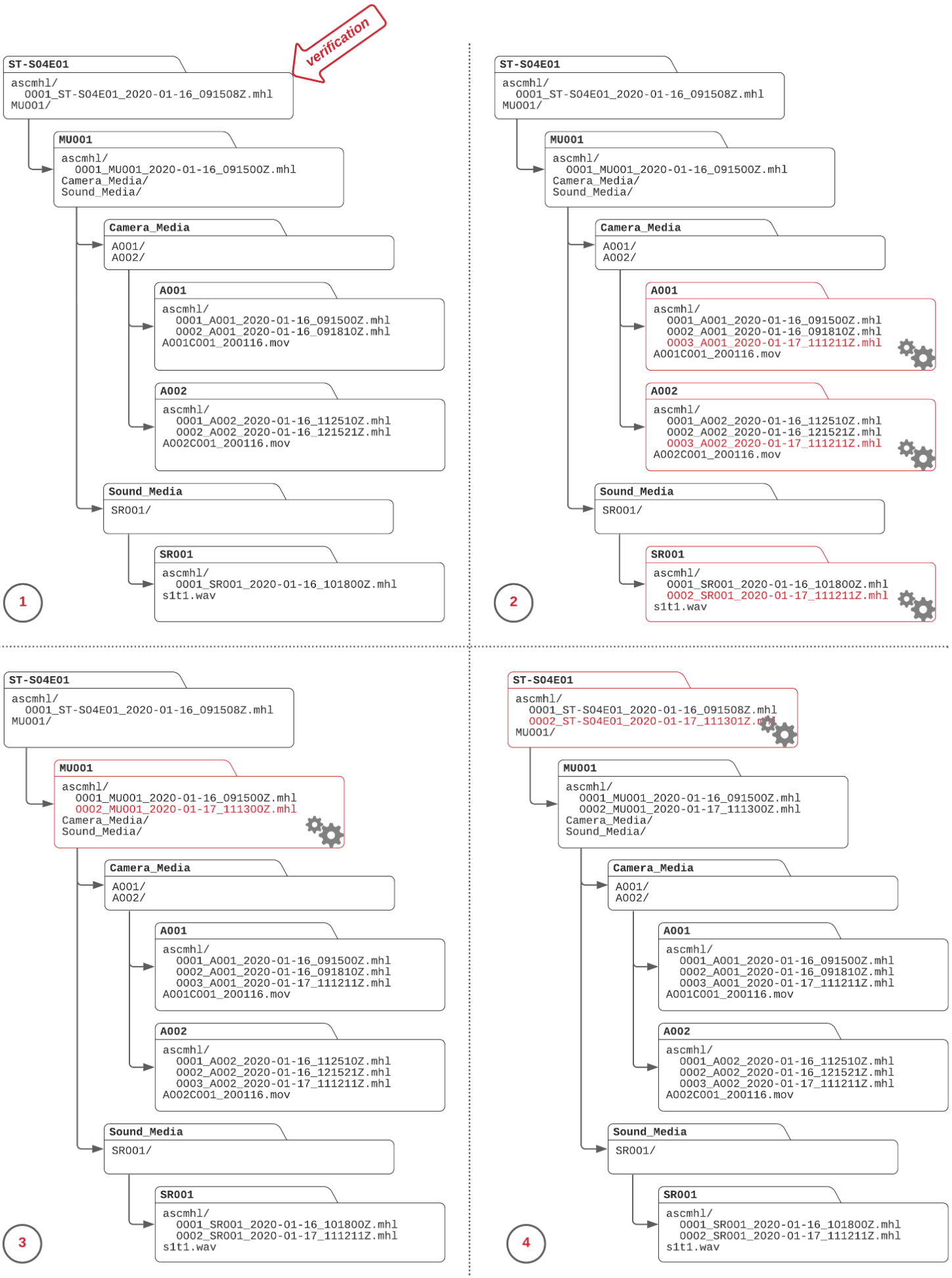
- Updates to a given ASC MHL History that affect the entire data set managed by that ASC MHL History, e.g. updates to ignore patterns (see IgnoreType) or running a verification, are also propagated into ASC MHL Histories nested within the given ASC MHL History. New generations of nested ASC MHL Histories are created and referenced in new generations up to the level of the given ASC MHL History.
- In contrast, updates to ASC MHL Histories do not propagate into higher-level ASC MHL Histories.

  Note: Since such updates are recorded in new ASC MHL Manifest generations in nested ASC MHL Histories, the references previously recorded in the higher level ASC MHL Histories are outdated following such update.

Example 1: Renaming a file in a subdirectory managed by a nested ASC MHL History does not affect additional ASC MHL Histories within the data set, as illustrated in the following diagram:



Example 2: Verifying a full data set with ASC MHL Histories on multiple directory levels leads to updates (i.e. new ASC MHL Manifest generations) in all (nested) ASC MHL Histories, as illustrated in the following diagram:


verification
ST-S04E01
ascmhl/
0001_ST-S04E01_2020-01-16_091508Z.mhl
MU001/
MU001
ascmhl/
0001_MU001_2020-01-16_091500Z.mhl
Camera_Media/
Sound_Media/
Camera_Media
A001/
A002/
A001
ascmhl/
0001_A001_2020-01-16_091500Z.mhl
0002_A001_2020-01-16_091810Z.mhl
A001C001_200116.mov
A002
ascmhl/
0001_A002_2020-01-16_112510Z.mhl
0002_A002_2020-01-16_121521Z.mhl
A002C001_200116.mov
Sound_Media
SR001/
SR001
ascmhl/
0001_SR001_2020-01-16_101800Z.mhl
s1t1.wav
1
ST-S04E01
ascmhl/
0001_ST-S04E01_2020-01-16_091508Z.mhl
MU001/
MU001
ascmhl/
0001_MU001_2020-01-16_091500Z.mhl
Camera_Media/
Sound_Media/
Camera_Media
A001/
A002/
A001
ascmhl/
0001_A001_2020-01-16_091500Z.mhl
0002_A001_2020-01-16_091810Z.mhl
0003_A001_2020-01-17_111211Z.mhl
A001C001_200116.mov
A002
ascmhl/
0001_A002_2020-01-16_112510Z.mhl
0002_A002_2020-01-16_121521Z.mhl
0003_A002_2020-01-17_111211Z.mhl
A002C001_200116.mov
Sound_Media
SR001/
SR001
ascmhl/
0001_SR001_2020-01-16_101800Z.mhl
0002_SR001_2020-01-17_111211Z.mhl
s1t1.wav
2
ST-S04E01
ascmhl/
0001_ST-S04E01_2020-01-16_091508Z.mhl
MU001/
MU001
ascmhl/
0001_MU001_2020-01-16_091500Z.mhl
0002_MU001_2020-01-17_111300Z.mhl
Camera_Media/
Sound_Media/
Camera_Media
A001/
A002/
A001
ascmhl/
0001_A001_2020-01-16_091500Z.mhl
0002_A001_2020-01-16_091810Z.mhl
0003_A001_2020-01-17_111211Z.mhl
A001C001_200116.mov
A002
ascmhl/
0001_A002_2020-01-16_112510Z.mhl
0002_A002_2020-01-16_121521Z.mhl
0003_A002_2020-01-17_111211Z.mhl
A002C001_200116.mov
Sound_Media
SR001/
SR001
ascmhl/
0001_SR001_2020-01-16_101800Z.mhl
0002_SR001_2020-01-17_111211Z.mhl
s1t1.wav
3
ST-S04E01
ascmhl/
0001_ST-S04E01_2020-01-16_091508Z.mhl
0002_ST-S04E01_2020-01-17_111301Z.mhl
MU001/
MU001
ascmhl/
0001_MU001_2020-01-16_091500Z.mhl
0002_MU001_2020-01-17_111300Z.mhl
Camera_Media/
Sound_Media/
Camera_Media
A001/
A002/
A001
ascmhl/
0001_A001_2020-01-16_091500Z.mhl
0002_A001_2020-01-16_091810Z.mhl
0003_A001_2020-01-17_111211Z.mhl
A001C001_200116.mov
A002
ascmhl/
0001_A002_2020-01-16_112510Z.mhl
0002_A002_2020-01-16_121521Z.mhl
0003_A002_2020-01-17_111211Z.mhl
A002C001_200116.mov
Sound_Media
SR001/
SR001
ascmhl/
0001_SR001_2020-01-16_101800Z.mhl
0002_SR001_2020-01-17_111211Z.mhl
s1t1.wav
4

5.3.3. Locating Hash Records

When parsing an ASC MHL History for information about a given file, that information may be distributed across one or more ASC MHL Manifests, or even ASC MHL Histories. Not every ASC MHL Manifest within an ASC MHL History necessarily covers all files/directories of the data set. At the same time, different ASC MHL Manifests may contain hash records for the same file/directory that were created using either the same, or a different hashing algorithm. An ASC MHL tool should use the ASC MHL Chain file to locate the ASC MHL Manifests of an ASC MHL History and parse them all to assemble the full set of hash records. The same mechanism shall be used for nested ASC MHL Histories. Even though a reference points to an individual ASC MHL Manifest, the entire ASC MHL History is parsed based on the entries in the ASC MHL Chain file.

ASC MHL implementations shall support the renaming mechanism specified in ASC MHL Rename, i.e. be able to locate hash records for files/directories that were renamed throughout the lifecycle of an ASC MHL History. The `previousPath` element (see HashType) is used to locate entries for a given file in previous ASC MHL Manifests.

5.4. ASC MHL Chain

An ASC MHL Chain is a file, as specified in ASC MHL Chain File, that serves as the table of contents of an ASC MHL History. It contains paths and file hashes for all ASC MHL Manifests contained in the ASC MHL History to allow verification of their integrity.

All ASC MHL Manifests in an ASC MHL Chain must have the same scope.

5.5. ASC MHL Collection

An ASC MHL Collection consists of an ASC MHL Collection file, as specified in ASC MHL Collection File, and all ASC MHL Manifests recorded in that ASC MHL Collection file.

ASC MHL Collection files and ASC MHL Chain files conform to the same XML Schema (i.e. they follow the same structure), however, ASC MHL Manifests in an ASC MHL Collection can have independent scopes. A collection can be used as a “packing list” or “receipt” and usually contains ASC MHL Manifest files that were created using the ASC MHL Flatten operation.

ASC MHL Manifest files are referenced in an ASC MHL Collection file using relative paths. These relative paths need to be maintained when moving files, to ensure the paths continue to resolve to the relative ASC MHL Manifest files.

Example: An ASC MHL Collection can be used to bundle ASC MHL Manifest files that cover separate camera cards. The ASC MHL Collection (alongside referenced ASC MHL Manifests) can then be sent, e.g. via email, to the recipient of the data delivery that contains the covered camera cards. Below diagram illustrates how ASC MHL Manifests from multiple camera cards (1) are copied to a common folder together with an ascmhl_collection.xml file (2) that contains the references to both ASC MHL Manifests contained in the ASC MHL Collection (3).

5.6. ASC MHL Operations

5.6.1. General

5.6.1.1. Operations

In addition to the structures and files introduced in previous sections, ASC MHL also specifies the conceptual operations, described in ASC MHL Operations, that implementations of ASC MHL shall support.

5.6.1.2. Ignore Semantic

Select files and directories (for instance, files that can change outside the control of the user) can be excluded (“ignored”) from ASC MHL operations. To ignore files/directories, “ignore patterns” are configured in ASC MHL tools and recorded in ASC MHL Manifests (see section IgnoreType and Appendix C: Syntax of the Ignore Pattern for details).

Ignore patterns apply to existing files/directories, as well as files/directories that may be added to the managed data set later (e.g. “.DS_Store” files).

ASC MHL Manifests shall not contain hash records for any files/directories that match an ignore pattern recorded in the ASC MHL Manifest. Any matching files/directories are also ignored when performing operations like ASC MHL Diff and ASC MHL Verify on the managed data set.

ASC MHL has a default set of ignore patterns to avoid common pitfalls. This default ignore pattern list is the equivalent of ignoring all files named `.DS_Store`, and ignoring all directories named `ascmhl`. Since these files and directories change by appending an ASC MHL History, no records for them can be included in ASC MHL Manifests.

5.6.1.3. Hash Algorithms

An ASC MHL History (and even a single ASC MHL Manifest) can include hash records with multiple hash algorithms, also called “hash formats”. This is to cover scenarios where hash values of multiple hash formats are required for different data management systems (see Appendix D for details on supported hash algorithms, their configuration, and encoding).

Users are free to change the hash algorithm for any hash record from one ASC MHL Manifest generation to the next. However, integrity verification needs to be performed based on a hash algorithm used to create previous hash records in the ASC MHL History.

5.6.2. ASC MHL Create

Create an ASC MHL Manifest or initiate an ASC MHL History based on input parameters that include:

- target directory (i.e. the scope of the ASC MHL Manifest/History)
- optional list of ignore patterns (see IgnoreType for details)
- hash algorithm(s)
- additional metadata to be recorded in (initial) ASC MHL Manifest

An ASC MHL Manifest is created with hashes for all files (including those located in subdirectory structures) within its scope, excluding files that either match the provided ignore pattern(s) or are already covered by existing (nested) ASC MHL Histories. Directory hashes can be included optionally. Hash values are computed using the specified hash algorithm(s).

While standalone ASC MHL Manifests are simply placed directly in the target directory, initiating an ASC MHL History includes the following additional steps:

- a directory `ascmhl` is created in the target directory and the initial ASC MHL Manifest is placed inside the `ascmhl` directory
- an ASC MHL Chain file is created inside the `ascmhl` directory with a record for the initial ASC MHL Manifest

Note 1: An ASC MHL Manifest can be created either as part of a “transfer” process (i.e. when managed data is copied, archived, restored, etc), or “in-place” process (i.e. ASC MHL Manifest is instantiated without copying/moving/altering the managed data), or “flatten” process (i.e. ASC MHL Manifest is created by the ASC MHL Flatten operation). Depending on the process used, the hashes recorded in an ASC MHL Manifest can stem from different sources, see ProcessType for details.

Note 2: While an ASC MHL History may be initiated “in-place” at any time, an ASC MHL History is typically initiated when an ASC MHL tool “transfers” (e.g. copy, move) a data set that does not contain an ASC MHL History.

Note 3: When creating an ASC MHL History for a data set that contains one or more subdirectories that are managed by existing ASC MHL Histories, the ASC MHL Manifest in the newly created ASC MHL History contains references to those nested ASC MHL Histories. It may also contain hash records for files outside of managed subdirectories.

5.6.3. ASC MHL Diff

The ASC MHL Diff operation can identify files within the scope of either an ASC MHL Manifest or ASC MHL History that are either:

- present in the file system, but neither recorded in the ASC MHL Manifest/History nor explicitly ignored, i.e. unknown files
- recorded in the ASC MHL Manifest/History, but not present in the file system, i.e. missing files

An implementation of ASC MHL can report both unknown and missing files within the scope of an ASC MHL Manifest/History.

Note: This operation does not verify the integrity of files and therefore doesn’t generate any hashes.

5.6.4. ASC MHL Verify

Verify the integrity of files covered by an ASC MHL History or ASC MHL Manifest. Users may choose to verify all files covered by an ASC MHL History/Manifest, or a select subset. This is achieved by:

- computing a hash for each specified file (using one of the algorithms recorded for that file in the ASC MHL History/Manifest) and
- comparing the computed hash against the recorded hash for that file.

Only hashes labeled as either `original` or `verified` can be used for verification. Hashes labeled as `failed` cannot be used. Verification is successful if the hashes match, and unsuccessful otherwise. Verification is also considered unsuccessful if a file is missing in the managed data set.

An implementation of ASC MHL shall report verification results for each file covered by the ASC MHL Manifest/History. When verifying a managed data set using an ASC MHL History, a new ASC MHL Manifest generation shall be appended to the ASC MHL History (see ASC MHL History Append).

Note 1: Hash records may be distributed across multiple ASC MHL Manifests within an ASC MHL History, see Locating Hash Records.

Note 2: Any intentional changes to a file (except renaming, see ASC MHL Rename) will result in failed verifications from the time of the change onward. This version of ASC MHL does not provide the means to rehabilitate intentionally changed files. Unintentional changes to a file can be resolved by replacing the file with a good copy that matches the original version.

5.6.5. ASC MHL History Append

Append an ASC MHL Manifest to an existing ASC MHL History. There are two general scenarios for this:

- a managed data set is verified and the results are recorded in a new ASC MHL Manifest generation
- files are added to a managed data set and a new ASC MHL Manifest generation is created with hashes for the added files

Verification of a managed data set can be triggered ad-hoc, but is usually done following a data management process that affects the managed data set, e.g. copying the managed data set to a new volume. In case of an ad-hoc verification, a user may choose to only verify a select subset of the managed data set. Results of a verification shall be recorded in a new ASC MHL Manifest for the files verified in the process.

When additional files are added to a managed data set using an ASC MHL tool, a new ASC MHL Manifest is created that only contains records for the newly added files.

The new ASC MHL Manifest follows the naming convention specified in Naming of the ASC MHL Manifest Files and is added to the ASC MHL Chain file as part of this operation.

Example 1: A single file is added to a managed data set. An ASC MHL Manifest is created with a hash record for the newly added file and appended to the ASC MHL History.

Example 2: A managed data set is copied to archive storage. On the archive storage volume, the data set is verified (see ASC MHL Manifest Verify) and an ASC MHL Manifest is created accordingly and appended to the ASC MHL History.

5.6.6. ASC MHL Rename

Rename files/folders within the scope of an ASC MHL History and record the previous and new file/folder paths in a newly created ASC MHL Manifest generation. Previous paths are recorded using the `previousPath` element, see HashType and DirectoryHashType. The `previousPath` element is recorded only in the hash record created as part of the renaming process and not repeated in subsequent ASC MHL Manifest generations, e.g. such created as part of an ASC MHL History Verify operation.

5.6.7. ASC MHL Flatten

Flatten/consolidate an ASC MHL History into a single ASC MHL Manifest. Hash records for a data set may be distributed across multiple ASC MHL Manifests within the ASC MHL History. This operation can be used to consolidate such distributed hash records into a standalone ASC MHL Manifest.

The flattened ASC MHL Manifest may cover the entire managed data set or just a (user-defined) subset. The resulting ASC MHL Manifest is not appended to the ASC MHL History and verification of data integrity is not part of the consolidation process.

The ASC MHL Flatten operation parses the entire ASC MHL History, including any nested ASC MHL Histories, and copies hash records into a new ASC MHL Manifest that are associated with files that are:

- part of the managed data set and
- selected by the user (default is all) and
- not excluded by the ignore semantic

Hash records that apply to directories are ignored by this operation, i.e. directory hashes are not consolidated into flattened ASC MHL Manifests. However, users may choose to re-create directory hashes for inclusion in the flattened ASC MHL Manifest as part of this operation, as long as no computation of actual file hashes is required (i.e. compatible hashes exist for all files covered by the directory hash).

Hashes labeled as `failed` are not transferred into the flattened ASC MHL Manifest. If multiple hashes of the same hashing algorithm exist for a single file, only the earliest entry is copied into the flattened ASC MHL Manifest.

Example 1: A history contains an `original` MD5 hash for a particular file, and more `verified` MD5 hashes for that file in subsequent ASC MHL Manifest generations. A flattened ASC MHL Manifest will only contain one MD5 hash for the particular file, labeled as `original`.

Example 2: A history contains an `original` MD5 hash for a particular file, and more `verified` C4 and XXH3 hashes for that file in subsequent ASC MHL Manifest generations. A flattened ASC MHL Manifest will contain the MD5 hash labeled as `original`, as well as one C4 and XXH3 hash each labeled as `verified`.

Any `previousPath` elements present in hash records are preserved in flattened ASC MHL Manifests.

Implementations may optionally consolidate `Metadata` elements present in hash records, e.g. by providing means to concatenate multiple values into a single entry in the flattened ASC MHL Manifest.

Attributes of the `HashType` `path` element are preserved based on the values associated with the latest hash record present in the ASC MHL History, in case more than one hash record exists for a given path.

Attributes of `HashFormatType` are preserved for each individual hash copied into the flattened ASC MHL Manifest.

Note: Flattened ASC MHL Manifests can be identified based on the `ProcessType` value `flatten` (see ProcessInfoType).

6. ASC MHL Manifest File

6.1. Schema

An ASC MHL Manifest File is an XML document, as specified in W3C XML 1.0, that consists of a single `hashlist` element (see section 5.2.2).

```
<schema targetNamespace="urn:ASC:MHL:v2.0" elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema" xmlns:ascmhl="urn:ASC:MHL:v2.0">
     <element name=”hashlist” type=”ascmhl:HashListType”>
</schema>
```

The namespace prefixes used in XML Schema definitions herein are not normative values and implementations shall perform correctly with any XML compliant prefix values.

6.2. Character Encoding

ASC MHL Manifests shall be encoded using the UTF-8 character encoding.

6.3. Naming of the ASC MHL Manifest Files

ASC MHL Manifest files that are part of an ASC MHL History shall conform to the following file naming convention:

<numbering> “_” <foldername> “_” <date> “_” <time> “.mhl”

<numbering> is a sequential number within the `ascmhl` folder in order of creation (4-digit, or more if required for values > 9999), starting with 1

<foldername> is the name of the folder that is covered by the ASC MHL Manifest (e.g. the name of the root directory)

<date> is the date of creation, using the format YYYY-MM-DD

<time> is the time of creation, using the format HHMMSSZ (trailing “Z” indicating “Zulu” time, see below)

Values for <date> and <time> are determined at the start of the operation that results in the creation of one or more ASC MHL Manifests and the values are represented using the Coordinated Universal Time (UTC) standard. E.g. all ASC MHL Manifest files that are created throughout (nested) ASC MHL Histories as the result of a verification process would share the

same date/time values even if time has passed between the creation of the individual files. If the verification process was started at 12:08:01 PM Pacific Time the value reflected in the file names would be `190801` (the UTC 24h representation of said time).

Example: `0001_A002R2EC_2019-01-07_080228Z.mhl`

6.4. Types

6.4.1. HashListType

```
<complexType name="HashListType">
     <element name="creatorinfo" type="ascmhl:CreatorInfoType"/>
     <element name="processinfo" type="ascmhl:ProcessInfoType"/>
     <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
     <element name="hashes" type="ascmhl:HashesType" minOccurs="0"/>
     <element name="references" type="ascmhl:ReferencesType" minOccurs="0"/>
     <attribute fixed="2.0" name="version" use="required"/>
</complexType>
```

Description:

`HashListType` is the complex type for the top-level `hashlist` element.

It requires the children `creatorInfo` and `processinfo`. At least one of the children `hashes` (a list of hashes) or `references` (a list of references to other ASC MHL documents) must be present. There is an additional, optional child `metadata` for carrying custom metadata concerning the entire hash list.

The `HashListType` `version` attribute shall be set to the value “2.0” (for distinguishing documents of this ASC MHL version 2 from the previous, incompatible MHL version 1).

6.4.2. CreatorInfoType

```
<complexType name="CreatorInfoType">
     <sequence>
          <element name="creationdate" type="dateTime"/>
          <element name="hostname" type="string"/>
          <element name="tool" type="ascmhl:ToolType"/>
          <element name="author" type="ascmhl:AuthorType" maxOccurs="unbounded"
minOccurs="0"/>
          <element name="location" type="string" minOccurs="0"/>
          <element name="comment" type="string" minOccurs="0"/>
     </sequence>
</complexType>
```

Description:

The `CreatorInfoType` contains information that applies to the entire ASC MHL Manifest.

Required child elements:

- `creationdate`: Date and time of the creation of the ASC MHL Manifest.
- `hostname`: Name of the computer used to create the ASC MHL Manifest.

- `tool`: Name and version of the software tool used to create the ASC MHL Manifest.

Optional child elements:

- `author`: Information about the person that initiated or controlled the process that created the ASC MHL Manifest document (multiple author elements are possible).
- `location`: Free form, human readable information about the geographical location where the process has been executed, e.g. “Los Angeles, CA”.
- `comment`: Informational, human readable text specifying a summary of the purpose, context, and parameters of the process.

6.4.3. ProcessInfoType

```
<complexType name="ProcessInfoType">
     <sequence>
          <element name="process" type="ascmhl:ProcessType"/>
          <element name="roothash" type="ascmhl:RootDirectoryHashType" minOccurs="0"/>
          <element name="ignore" type="ascmhl:IgnoreType" minOccurs="0"/>
     </sequence>
</complexType>
```

Description:

The `ProcessInfoType` contains information that applies to the entire ASC MHL Manifest.

Required child elements:

- `process`: Type of process used when the ASC MHL Manifest was created.

Optional child elements:

- `roothash`: Contains hash values representing the entire managed data set.
- `ignore`: Element specifying which file patterns have been used to ignore files.

6.4.4. AuthorType

```
<complexType name="AuthorType">
     <simpleContent>
          <extension base="string">
               <attribute name="email" type="ascmhl:EmailAddressAttributeType"/>
               <attribute name="phone" type="string"/>
               <attribute name="role"  type="string"/>
          </extension>
     </simpleContent>
</complexType>
```

Description:

`AuthorType` contains information about the person that initiated or controlled the process that created the ASC MHL Manifest.

The value of the `author` element is the name of the person. The optional `email` and `phone` attributes can be used to add contact information for the person. The optional `role` attribute can be used to distinguish the different roles of the persons when adding multiple author elements.

6.4.5. ToolType

```
<complexType name="ToolType">
     <simpleContent>
          <extension base="string">
               <attribute name="version" type="string"/>
          </extension>
     </simpleContent>
</complexType>
```

Description:

`ToolType` contains name and version information about the software tool or system used to create the ASC MHL Manifest.

The value of the `tool` element is the name of the software. The optional `version` attribute can be used to specify the version of the software.

6.4.6. ProcessType

```
<simpleType name="ProcessType">
     <restriction base="string">
          <enumeration value="in-place"/>
          <enumeration value="transfer"/>
          <enumeration value="flatten"/>
     </restriction>
</simpleType>
```

Description:

`ProcessType` carries information about the data management process during which the ASC MHL Manifest was created.

It can have the values `in-place`, `transfer`, or `flatten`:

- An `in-place` process doesn't move or duplicate any files, it only creates the ASC MHL Manifest representing a new generation. This can be useful when, for example, a copy has been made with a non-ASC MHL-aware system (such as Finder or Explorer) and the ASC MHL Manifest is created afterwards "in-place". Hashes are computed directly from the files referenced in the ASC MHL Manifest.
- A `transfer` process is a process that copied, archived, restored, or otherwise created a new instance of the files and at the same time (e.g. without extra user interaction) created the ASC MHL Manifest representing a new generation. Hashes are computed directly from files at the source and/or destination and the ASC MHL tool creating the ASC MHL Manifest guarantees that the files on the source and destination are identical, either by comparing hashes created from files at both the source and destination or other bona fide means.

- A `flatten` process indicates that the ASC MHL Manifest was created using the ASC MHL Flatten operation (see ASC MHL Flatten), i.e. hash values were not computed but instead copied from existing ASC MHL Manifest files.

6.4.7. IgnoreType

```
<complexType name="IgnoreType">
     <sequence>
          <element name="pattern" type="string" maxOccurs="unbounded"/>
     </sequence>
</complexType>
```

Description:

`IgnoreType` contains patterns that specify which files have been ignored during the creation of the ASC MHL Manifest.

The syntax and semantics of the pattern value is equivalent to one line of a `.gitignore` file as specified in the git reference (https://git-scm.com/docs/gitignore). See Appendix C for the detailed specification.

6.4.8. RelativePathType

```
<simpleType name="RelativePathType">
     <restriction base="string"/>
</simpleType>
```

Description:

`RelativePathType` is a string describing a path in the file system, relative to the scope of an ASC MHL Manifest. The following restrictions apply to values of this type:

- case shall be preserved
- whitespaces shall be preserved
- forward slash (“/”) shall be used to separate components (i.e. directory levels)
- the path shall not start with forward slash
- the most direct path shall be used, e.g. “..” path segments are not allowed

6.4.9. HashesType

```
<complexType name="HashesType">
     <choice minOccurs="1" maxOccurs="unbounded">
            <element name="hash" type="ascmhl:HashType"/>
            <element name="directoryhash" type="ascmhl:DirectoryHashType"/>
     </choice>
</complexType>
```

Description:

`HashesType` contains a list of one or more `hash` and/or `directoryhash` child elements.

No two `hash` or `directoryhash` child elements in an element of type `HashesType` shall have identical values for `path`.

6.5. HashType

```
<complexType name="HashType">
     <sequence>
          <element name="path">
               <complexType>
                    <simpleContent>
                         <extension base="ascmhl:RelativePathType">
                              <attribute name="size" type="integer"/>
                              <attribute name="creationdate" type="dateTime"/>
                              <attribute name="lastmodificationdate" type="dateTime"/>
                         </extension>
                    </simpleContent>
               </complexType>
          </element>
          <sequence>
               <element name="c4" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="md5" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="sha1" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh128" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh3" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh64" type="ascmhl:HashFormatType" minOccurs="0"/>
          </sequence>
          <element name="previousPath" type="ascmhl:RelativePathType" minOccurs="0"/>
          <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
     </sequence>
</complexType>
```

Description:

`HashType` contains a path, a number of hashes, and optionally additional metadata for a file in the file system.

Required child elements:

- `path`: Path to the file based on which the hash values were created. The `path` element can have three optional parameters:
    - `lastmodificationdate`: The last modification date of the associated file in the file system.
    - `creationdate`: The creation date of the associated file in the file system.
    - `size`: The size of the associated file.
- One or more of the following elements shall be present: `md5`, `sha1`, `c4`, `xxh64`, `xxh3`, `xxh128`.

Optional child elements:

- `previousPath`: files within a managed data set may be renamed throughout the lifecycle of an ASC MHL History. If a file name changed from it's most recent record in the ASC

MHL History, the `previousPath` element carries the former value in the `path` element, i.e. the path that was used in the previous ASC MHL Manifest within the ASC MHL History.

Example:

- An ASC MHL Manifest contains the following `hash` element:

```
<hash>
     <path size="5" lastmodificationdate="2019-10-11T15:56:03+02:00">
       Clips/0001.mov</path>
     <xxh64 action="original">7680e5f98f4a80fd</xxh64>
</hash>
```

- The associated file is renamed using an ASC MHL tool and a new ASC MHL Manifest is appended to the ASC MHL History with the following entry:

```
<hash>
     <path size="5" lastmodificationdate="2019-10-11T16:15:03+02:00">
      Clips/A002C007_141024.mov</path>
     <xxh64 action="verified">7680e5f98f4a80fd</xxh64>
     <previousPath>Clips/0001.mov</previousPath>
  </hash>
```

- `metadata`: Custom metadata concerning the file.

6.5.1. HashFormatType

```
<simpleType name="ActionAttributeType">
     <restriction base="string">
          <enumeration value="original"/>
          <enumeration value="verified"/>
          <enumeration value="failed"/>
     </restriction>
</simpleType>
<complexType name="HashFormatType">
     <simpleContent>
          <extension base="string">
               <attribute name="action" type="ascmhl:ActionAttributeType"/>
               <attribute name="hashdate" type="dateTime"/>
          </extension>
     </simpleContent>
</complexType>
```

Description:

`HashFormatType` is the complex type for the `md5`, `sha1`, `c4`, `xxh64`, `xxh3`, and `xxh128` children of the `hash` element.

The file hash value of the element is the hash value of the data content of the file in the encoding specified by the individual hash format per Appendix D.

The action attribute:

If the parent `hash` element describes a file, the elements of type `HashFormatType` are required to have the `action` attribute that qualifies each element. Possible values of the attribute are:

- `original` indicates that the hash is the initial hash for a file or directory within the ASC MHL History.
- `verified` indicates that the hash was generated after a file was successfully verified against a previously recorded hash in the ASC MHL History. The verification is deemed successful when a hash generated using one of the algorithms recorded in the ASC MHL History matches the corresponding hash in the ASC MHL History.
  Note: While the verification is based on a previously used algorithm, the hashes recorded in the new ASC MHL Manifest generation can be of different types and may or may not include a hash of the previous type.
- `failed` indicates that the hash value has been created from a file but verification against a previously recorded hash in the ASC MHL History failed, i.e. the file was altered.

The hashdate attribue:

The elements of type `HashFormatType` can have a `hashdate` attribute specifying the date and time of

- the computation of a file hash, or
- the creation of a directory hash.

When the `hashdate` attribute is missing it is assumed that the hash has been created as part of the process that created the ASC MHL Manifest (and thus at roughly the `creationdate` date and time of the `creatorinfo` element).

See Appendix D for details on configuration and encoding of hash formats.

6.5.2. DirectoryHashType

```
<complexType name="DirectoryHashType">
        <sequence>
            <element name="path">
                <complexType>
                    <simpleContent>
                        <extension base="ascmhl:RelativePathType">
                            <attribute name="creationdate" type="dateTime"/>
                            <attribute name="lastmodificationdate" type="dateTime"/>
                        </extension>
                    </simpleContent>
                </complexType>
            </element>
            <element name="content" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="structure" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="previousPath" type="ascmhl:RelativePathType" minOccurs="0"/>
            <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
        </sequence>
    </complexType>
```

Description:

`DirectoryHashType` contains a `path` element and `content` and `structure` hashes for a directory.

Required child elements:

- `path`: Relative path to the directory represented by the `content` and `structure` hashes. The `path` element can have two optional parameters:
  - `lastmodificationdate`: The last modification date of the associated directory in the file system.
  - `creationdate`: The creation date of the associated directory in the file system.
- `content`: A directory hash value created as follows:
  1. For the immediate children of a directory specified through the `path` element, collect a list of hashes containing
     - all file hashes (for files) and
     - the content directory hashes (for directories)
  2. A hash value is created from the list of hashes, following the process described in Appendix G.
- `structure`: A directory hash value created as follows:
  1. For the immediate children of a directory specified through the `path` element, collect a list of hashes where each hash is computed as follows:
     - A file's encoded name is concatenated with it's encoded hash, then a hash value is computed from those concatenated bytes. The resulting hash value is appended to the list of hashes.
     - A directory's encoded name is concatenated with it's encoded structure hash, then a hash value is computed from those concatenated bytes. The resulting hash value is appended to the list of hashes.
  2. A hash value is created from the list of hashes, following the process described in Appendix G.

Optional child elements:

- `previousPath`: directories within a managed data set may be renamed throughout the lifecycle of an ASC MHL History. If a directory name changed from it's most recent record in the ASC MHL History, the `previousPath` element carries the former value in the `path` element, i.e. the path that was used in the previous ASC MHL Manifest within the ASC MHL History.

  Example:

  - An ASC MHL Manifest contains the following `directoryhash` element:

```
<directoryhash>
      <path>Clips_0001</path>
  <content>
      <xxh64 action="original">7680e5f98f4a80fd</xxh64>
  </content>
  <structure>
      <xxh64 action="original">8f4a80fd7680e5f9</xxh64>
```

```
</structure>
</directoryhash>
```

- The associated directory is renamed using an ASC MHL tool and a new ASC MHL Manifest is appended to the ASC MHL History with the following entry:

```
<directoryhash>
      <path>Clips_141024</path>
      <content>
           <xxh64 action="verified">7680e5f98f4a80fd</xxh64>
      </content>
      <structure>
           <xxh64 action="verified">8f4a80fd7680e5f9</xxh64>
      </structure>
      <previousPath>Clips_0001</previousPath>
 </directoryhash>
```

- `metadata`: Custom metadata concerning the directory.

The `content` and `structure` elements have the same number of children and `HashFormatType` children of both the `content` and `structure` elements must represent the same hash formats. Example:

```
<content>
      <md5 action="verified">afd3e0ec44cdcde02d17b580329b566b</md5>
      <xxh64 action="verified">7680e5f98f4a80fd</xxh64>
</content>
<structure>
      <md5 action="verified">4cdcde02d17b580329b566bafd3e0ec4</md5>
      <xxh64 action="verified">8f4a80fd7680e5f9</xxh64>
</structure>
```

Note 1: If a directory child of a folder is the root folder of a nested ASC MHL History, the `content` and `structure` hashes of that folder can be taken from the latest available `roothash` element of the nested ASC MHL History, if that roothash element contains appropriate directory hashes.

Note 2: All hash values used to create the content and structure directory hash values must be of the individual hash format of the `HashFormatType` element. The content and the structure directory hash values are encoded by the individual hash format per Appendix D.

6.5.3. RootDirectoryHashType

```
<complexType name="RootDirectoryHashType">
        <sequence>
            <element name="content" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="structure" type="ascmhl:DirectoryHashFormatContainerType"/>
        </sequence>
    </complexType>
```

Description:

See definitions for `content` and `structure` hashes in DirectoryHashType.

Note: Specifying a path for root hashes is impractical, but requiring the path element for DirectoryHashType is desirable to allow for easy XML Schema validation of hash entries. This additional type was therefore introduced specifically for root hashes.

6.5.4. DirectoryHashFormatContainerType

```
<complexType name="DirectoryHashFormatContainerType">
     <sequence>
          <element name="c4" type="ascmhl:HashFormatType" minOccurs="0"/>
          <element name="md5" type="ascmhl:HashFormatType" minOccurs="0"/>
          <element name="sha1" type="ascmhl:HashFormatType" minOccurs="0"/>
          <element name="xxh128" type="ascmhl:HashFormatType" minOccurs="0"/>
          <element name="xxh64" type="ascmhl:HashFormatType" minOccurs="0"/>
          <element name="xxh3" type="ascmhl:HashFormatType" minOccurs="0"/>
     </sequence>
 </complexType>
```

The `DirectoryHashFormatContainerType` contains one or more child elements `md5`, `sha1`, `c4`, `xxh3`, `xxh64` and/or `xxh128`.

6.5.5. ReferencesType and HashListReferenceType

```
<complexType name="ReferencesType">
     <sequence>
          <element maxOccurs="unbounded" name="hashlistreference"
                type="ascmhl:HashListReferenceType"/>
     </sequence>
</complexType>
<complexType name="HashListReferenceType">
     <sequence>
          <element name="path" type="ascmhl:RelativePathType"/>
          <element name="c4" type="ascmhl:HashFormatType"/>
     </sequence>
</complexType>
```

Description:

`ReferencesType` is the complex type for the `references` child of the top-level `hashlist` element. The `HashListReferenceType` is the complex type for the `hashlistreference` children of the `references` element.

Required child elements:

- `path`: Path to the referenced ASC MHL Manifest, relative to the scope of the current ASC MHL Manifest.
- `c4`: A hash value of the referenced ASC MHL Manifest in the C4 format.

6.5.6. MetadataType

```
<complexType name="MetadataType">
     <complexContent>
          <extension base="anyType"/>
```

```
    </complexContent>
</complexType>
```

Description:

`MetadataType` is the complex type for the `metadata` children of the `hashlist` element and the `hash` elements.

As a child of the `hashlist` element it can include information concerning the entire hash list. As a child of a `hash` element it can include information concerning a file or directory. The `metadata` element can contain any custom attributes or elements, e.g. free text, key-value pairs, or entire XML structures. The content of a `metadata` element of course must be formatted to maintain the validity of the XML document.

7. ASC MHL Chain File

7.1. Schema

An ASC MHL Chain file is an XML document, as specified in W3C XML 1.0, that consists of a single `ascmhldirectory` element.

```
<schema targetNamespace="urn:ASC:MHL:DIRECTORY:v2.0" elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ascmhldirectory="urn:ASC:MHL:DIRECTORY:v2.0">
  <element name=”ascmhldirectory” type=”ascmhl:DirectoryType”>
</schema>
```

The namespace prefixes used in XML Schema definitions herein are not normative values and implementations shall perform correctly with any XML compliant prefix values.

7.2. Character Encoding

ASC MHL Chain files shall be encoded using the UTF-8 character encoding.

7.3. Naming of the ASC MHL Chain Files

ASC MHL Chain files shall be named `ascmhl_chain.xml`

7.4. ASC MHL Chain XML Format

7.4.1. DirectoryType

```
<complexType name="DirectoryType">
    <sequence>
        <element name="hashlist" type="ascmhldirectory:HashlistType"
maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Description:

`DirectoryType` contains a list of one or more `HashlistType` child elements.

No two `hashlist` child elements of an element of type `DirectoryType` shall have identical values for their `sequencenr` attribute. The `sequencenr` attribute shall be a continuous count that starts at 1 for the first `hashlist` child element and is incremented by one for each subsequent entry.

7.4.2. HashlistType

```
<complexType name="HashlistType">
     <attribute name="sequencenr" type="integer"/>
     <sequence>
          <element name="path" type="ascmhl:RelativePathType"/>
          <element name="c4" type="ascmhl:HashFormatType"/>
     </sequence>
</complexType>
```

Description:

Required child elements:

- `path`: Path to the referenced ASC MHL Manifest relative to the ASC MHL Chain file.
- `c4`: A hash value of the referenced ASC MHL Manifest in the C4 format.

The `sequencenr` attribute shall be present and uniquely identify the element within the document.

8. ASC MHL Collection File

8.1. Schema

An ASC MHL Collection file is an XML document, as specified in W3C XML 1.0, that consists of a single `ascmhldirectory` element.

```
<schema targetNamespace="urn:ASC:MHL:DIRECTORY:v2.0" elementFormDefault="qualified"
     xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:ascmhldirectory="urn:ASC:MHL:DIRECTORY:v2.0">
  <element name=”ascmhldirectory” type=”ascmhl:DirectoryType”>
</schema>
```

The namespace prefixes used in XML Schema definitions herein are not normative values and implementations shall perform correctly with any XML compliant prefix values.

8.2. Character Encoding

ASC MHL Collection files shall be encoded using the UTF-8 character encoding.

8.3. Naming of the ASC MHL Collection Files

ASC MHL Collection files shall be named `ascmhl_collection.xml`

Appendix

Appendix A: ASC MHL Manifest XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:ASC:MHL:v2.0" elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema" xmlns:ascmhl="urn:ASC:MHL:v2.0">
    <simpleType name="EmailAddressAttributeType">
        <restriction base="string">
            <pattern value="[^@]+@[^\.]+\..+"/>
        </restriction>
    </simpleType>
    <simpleType name="ActionAttributeType">
        <restriction base="string">
            <enumeration value="original"/>
            <enumeration value="verified"/>
            <enumeration value="failed"/>
        </restriction>
    </simpleType>
    <simpleType name="RelativePathType">
        <restriction base="string"/>
    </simpleType>
    <complexType name="HashListType">
     <sequence>
        <element name="creatorinfo" type="ascmhl:CreatorInfoType"/>
        <element name="processinfo" type="ascmhl:ProcessInfoType"/>
        <element name="hashes" type="ascmhl:HashesType"/>
        <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
        <element name="references" type="ascmhl:ReferencesType" minOccurs="0"/>
     </sequence>
        <attribute fixed="2.0" name="version" use="required"/>
    </complexType>
    <complexType name="CreatorInfoType">
        <sequence>
            <element name="creationdate" type="dateTime"/>
            <element name="hostname" type="string"/>
            <element name="tool" type="ascmhl:ToolType"/>
            <element name="author" type="ascmhl:AuthorType" maxOccurs="unbounded"
minOccurs="0"/>
            <element name="location" type="string" minOccurs="0"/>
            <element name="comment" type="string" minOccurs="0"/>
        </sequence>
    </complexType>
    <complexType name="ProcessInfoType">
        <sequence>
            <element name="process" type="ascmhl:ProcessType"/>
            <element name="roothash" type="ascmhl:RootDirectoryHashType"
minOccurs="0"/>
            <element name="ignore" type="ascmhl:IgnoreType" minOccurs="0"/>
        </sequence>
    </complexType>
    <complexType name="AuthorType">
        <simpleContent>
            <extension base="string">
                <attribute name="email" type="ascmhl:EmailAddressAttributeType"/>
                <attribute name="phone" type="string"/>
```

```
            <attribute name="role" type="string"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="ToolType">
    <simpleContent>
        <extension base="string">
            <attribute name="version" type="string"/>
        </extension>
    </simpleContent>
</complexType>
<simpleType name="ProcessType">
    <restriction base="string">
        <enumeration value="in-place"/>
        <enumeration value="transfer"/>
       <enumeration value="flatten"/>
    </restriction>
</simpleType>
   <complexType name="IgnoreType">
    <sequence>
        <element name="pattern" type="string" maxOccurs="unbounded"/>
    </sequence>
</complexType>
<complexType name="HashesType">
    <choice minOccurs="1" maxOccurs="unbounded">
        <element name="hash" type="ascmhl:HashType"/>
        <element name="directoryhash" type="ascmhl:DirectoryHashType"/>
    </choice>
</complexType>
<complexType name="HashType">
    <sequence>
        <element name="path">
            <complexType>
                <simpleContent>
                    <extension base="ascmhl:RelativePathType">
                        <attribute name="size" type="integer"/>
                        <attribute name="creationdate" type="dateTime"/>
                        <attribute name="lastmodificationdate" type="dateTime"/>
                    </extension>
                </simpleContent>
            </complexType>
        </element>
        <sequence>
            <element name="c4" type="ascmhl:HashFormatType" minOccurs="0"/>
            <element name="md5" type="ascmhl:HashFormatType" minOccurs="0"/>
            <element name="sha1" type="ascmhl:HashFormatType" minOccurs="0"/>
            <element name="xxh128" type="ascmhl:HashFormatType" minOccurs="0"/>
            <element name="xxh3" type="ascmhl:HashFormatType" minOccurs="0"/>
            <element name="xxh64" type="ascmhl:HashFormatType" minOccurs="0"/>
        </sequence>
        <element name="previousPath" type="ascmhl:RelativePathType" minOccurs="0"/>
        <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
    </sequence>
</complexType>
 <complexType name="DirectoryHashType">
    <sequence>
        <element name="path">
            <complexType>
                <simpleContent>
```

```
                        <extension base="ascmhl:RelativePathType">
                            <attribute name="creationdate" type="dateTime"/>
                            <attribute name="lastmodificationdate" type="dateTime"/>
                        </extension>
                    </simpleContent>
                </complexType>
            </element>
            <element name="content" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="structure" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="previousPath" type="ascmhl:RelativePathType" minOccurs="0"/>
            <element name="metadata" type="ascmhl:MetadataType" minOccurs="0"/>
        </sequence>
    </complexType>
<complexType name="RootDirectoryHashType">
        <sequence>
            <element name="content" type="ascmhl:DirectoryHashFormatContainerType"/>
            <element name="structure" type="ascmhl:DirectoryHashFormatContainerType"/>
        </sequence>
    </complexType>
    <complexType name="HashFormatType">
        <simpleContent>
            <extension base="string">
                <attribute name="action" type="ascmhl:ActionAttributeType"/>
                <attribute name="hashdate" type="dateTime"/>
            </extension>
        </simpleContent>
    </complexType>
     <complexType name="DirectoryHashFormatContainerType">
          <sequence>
               <element name="c4" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="md5" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="sha1" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh128" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh3" type="ascmhl:HashFormatType" minOccurs="0"/>
               <element name="xxh64" type="ascmhl:HashFormatType" minOccurs="0"/>
          </sequence>
    </complexType>
    <complexType name="ReferencesType">
        <sequence>
            <element name="hashlistreference"
                type="ascmhl:HashListReferenceType" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="HashListReferenceType">
        <sequence>
            <element name="path" type="ascmhl:RelativePathType"/>
            <element name="c4" type="ascmhl:HashFormatType"/>
        </sequence>
    </complexType>
    <complexType name="MetadataType">
        <complexContent>
            <extension base="anyType"/>
        </complexContent>
    </complexType>
    <element name="hashlist" type="ascmhl:HashListType"/>
</schema>
```

Appendix B: Example ASC MHL XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<hashlist version="2.0"
  xmlns="urn:ASC:MHL:v2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ASC:MHL:v2.0 ASCMHL.xsd">
  <creatorinfo>
    <creationdate>2019-10-11T15:56:03+02:00</creationdate>
    <hostname>donkey.local</hostname>
    <tool version="0.0.3">ascmhl-tool</tool>
    <author email="foo@bar.foo.foo.com" phone="+1 234 567 890 88">Liz Foo</author>
    <author email="zed@bar.foo.foo.com" phone="+1 234 567 890 99">Ben Zed</author>
    <location>Munich, Germany</location>
  </creatorinfo>
  <processinfo>
    <process>in-place</process>
    <roothash>
      <content>
        <xxh64>7680e5f98f4a80fd</xxh64>
      </content>
      <structure>
        <xxh64>8f4a80fd7680e5f9</xxh64>
      </structure>
    </roothash>
    <ignore>
      <pattern>*.DS_Store</pattern>
      <pattern>/tmp</pattern>
    </ignore>
  </processinfo>
  <hashes>
    <hash>
      <path size="5"
lastmodificationdate="2019-10-11T15:56:03+02:00">Clips/A002C006_141024_R2EC.mov</path>
      <xxh64 action="original">0ea03b369a463d9d</xxh64>
    </hash>
    <hash>
      <path size="5"
lastmodificationdate="2019-10-11T15:56:03+02:00">Clips/A002C007_141024_R2EC.mov</path>
      <xxh64 action="original">7680e5f98f4a80fd</xxh64>
    </hash>
    <directoryhash>
      <path lastmodificationdate="2019-10-11T15:56:01+02:00">Clips/</path>
          <content>
               <xxh64>7680e5f98f4a80fd</xxh64>
          </content>
          <structure>
               <xxh64>8f4a80fd7680e5f9</xxh64>
          </structure>
      <metadata>test2</metadata>
    </directoryhash>
    <hash>
      <path size="58"
lastmodificationdate="2019-10-11T15:56:03+02:00">Sidecar.txt</path>
      <xxh64 action="failed">3ab5a4166b9bde44</xxh64>
      <metadata bar="Foo"/>
    </hash>
```

```
  </hashes>
  <metadata>
    test
    <foo bar="1">lorem</foo>
  </metadata>
  <references>
    <hashlistreference>
      <path>A002R2EC/ascmhl/0002_A002R2EC_2020-01-17_143000.mhl</path>
<c4>c418T9ncneEGEMT5NopfHGRPRBAuoxwYkP6w5S8xChNBYnFMZ4AknVurpwxTPsUouLtF9NGyxMBsZTDBEEp
L3JdHoG</c4>
    </hashlistreference>
    <hashlistreference>
      <path>A003R2EC/ascmhl/0002_A003R2EC_2020-01-17_143000.mhl</path>
<c4>c418T9ncneEGEMT5NopfHGRPRBAuoxwYkP6w5S8xChNBYnFMZ4AknVurpwxTPsUouLtF9NGyxMBsZTDBEEp
L3JdHoG</c4>
    </hashlistreference>
  </references>
</hashlist>
```

Appendix C: Syntax of the Ignore Pattern

Adapted from https://git-scm.com/docs/gitignore, one line in the .gitignore file format equals the value of one “pattern” element.

- Trailing spaces are ignored unless they are quoted with backslash (“\”).
- An optional prefix `"!"` which negates the pattern; any matching file excluded by a previous pattern will become included again. It is not possible to re-include a file if a parent directory of that file is excluded. Git doesn’t list excluded directories for performance reasons, so any patterns on contained files have no effect, no matter where they are defined. Put a backslash (`"\"`) in front of the first `"!"` for patterns that begin with a literal `"!"`, for example, “`\!important!.txt`".
- The slash / is used as the directory separator. Separators may occur at the beginning, middle or end of the search pattern.
- If there is a separator at the beginning or middle (or both) of the pattern, then the pattern is relative to the scope of the ASC MHL Manifest. Otherwise the pattern may also match at any level below the root path.
- If there is a separator at the end of the pattern then the pattern will only match directories, otherwise the pattern can match both files and directories.
- For example, a pattern `doc/frotz/` matches `doc/frotz` directory, but not `a/doc/frotz directory`; however `frotz/` matches `frotz` and `a/frotz` that is a directory (all paths are relative to the root path).
- An asterisk `"*"` matches anything except a slash. The character `"?"` matches any one character except `"/"`. The range notation, e.g. `[a-zA-Z]`, can be used to match one of the characters in a range.

Two consecutive asterisks ("**") in patterns matched against full pathname may have special meaning:

- A leading "**" followed by a slash means match in all directories. For example, "**/foo" matches file or directory "foo" anywhere, the same as pattern "foo". "**/foo/bar" matches file or directory "bar" anywhere that is directly under directory “foo".
- A trailing "/**" matches everything inside. For example, "abc/**" matches all files inside directory "abc", relative to the root path, with infinite depth.
- A slash followed by two consecutive asterisks then a slash matches zero or more directories. For example, "a/**/b" matches "a/b", "a/x/b", "a/x/y/b" and so on.
- Other consecutive asterisks are considered regular asterisks and will match according to the previous rules.

Appendix D: Hash Format Configuration and Encoding

MD5

The MD5 digest algorithm [RFC1321] takes no explicit parameters. An MD5 digest is a 128-bit string in base64 [RFC4648] encoding viewed as a 16-octet octet stream (32 characters).

Example: `40e52b717fab6af085566c769cd9c6ea`

SHA1

The SHA-1 digest algorithm [RFC3174] takes no explicit parameters. A SHA-1 digest is a 160-bit string in base64 [RFC4648] encoding viewed as a 16-octet octet stream (40 characters).

Example: `48146b795eab757a46261b722bf0d8312e113ff8`

C4

The C4 digest algorithm [SMPTE ST 2114] is a SHA-512 digest encoded in Base58 prepended by the string “c4” (90 characters).

Example:
`c4137gLfVPqPdRgKr8yXNp1CRgJwSN9Yr4Sh3aaVT6uGmN5GgA7HkHcrTmQxy4t2ZmfWYMqQmF4u7ZRMKtPUM965n2`

XXH64

The XXH64 digest algorithm [xxHash] takes a seed value as an input parameter. For ASC MHL the seed value is 0 (zero) and the implementation is big endian. The 64-bit digest value is viewed as a 16-octet octet stream (32 characters).

Example: `bca8c0744bf7f78d`

XXH3 and XXH128

The two XXH3 digest algorithms [xxHash] take a seed value as an input parameter. For ASC MHL the seed value is 0 (zero) and the implementation is big endian. The 64-bit digest value is viewed as a 16-octet octet stream (32 characters), the 128-bit digest value is viewed as a 32-octet octet stream (64 characters).

Example (XXH3): `3ab5a4166b9bde44`

Example (XXH128): `00fd03cd9996ee8cf8be6a756bf82a42`

Appendix E: ASC MHL Directory XML Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<schema targetNamespace="urn:ASC:MHL:DIRECTORY:v2.0"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ascmhldirectory="urn:ASC:MHL:DIRECTORY:v2.0"
    xmlns:ascmhl="urn:ASC:MHL:v2.0"
    elementFormDefault="qualified">
    <import
schemaLocation="https://raw.githubusercontent.com/ascmitc/mhl/master/xsd/ASCMHL.xsd"
namespace="urn:ASC:MHL:v2.0"/>
    <complexType name="DirectoryType">
        <sequence>
            <element name="hashlist" type="ascmhldirectory:HashlistType"
maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="HashlistType">
     <sequence>
        <element name="path" type="ascmhl:RelativePathType"/>
        <element name="c4" type="ascmhl:HashFormatType"/>
     </sequence>
        <attribute name="sequencenr" type="integer"/>
    </complexType>
    <element name="ascmhldirectory" type="ascmhldirectory:DirectoryType"/>
</schema>
```

Appendix F: Example ASC MHL Chain File

```
<?xml version="1.0" encoding="UTF-8"?>
<ascmhldirectory xmlns="urn:ASC:MHL:DIRECTORY:v2.0">
     <hashlist sequencenr="1">
          <path>0001_A002R2EC_2020-01-16_091500.mhl</path>
          <c4>c418T9ncneEGEMT5NopfHGRPRBAuoxwYkP6w5S8xChNBYnFMZ4AknVurpwxTPsUouL
               tF9NGyxMBsZTDBEEpL3JdHoG</c4>
     </hashlist>
     <hashlist sequencenr="2">
          <path>0002_A002R2EC_2020-01-16_091500.mhl</path>
          <c4>c418T9ncneEGEMT5NopfHGRPRBAuoxwYkP6w5S8xChNBYnFMZ4AknVurpwxTPsUouL
               tF9NGyxMBsZTDBEEpL3JdHoG</c4>
     </hashlist>
</ascmhldirectory>
```

Appendix G: Process for creating a hash of hashes.

To produce a hash of hashes, all child hashes included in the computation of the parent hash must be of the same checksum algorithm type. For example, an MD5 directory hash can only be computed from a list of child MD5 checksums.

The process to compute a hash from a list of hashes is as follows:

1. Given a list of hashes, reset the hash-generator* to a clear (empty / new) state.
2. Sort the list of hashes by hash value lexicographically.
3. For each hash in the list of hashes:
    a. Encode the hash value into its appropriate byte representation as specified in Appendix D.
    b. Write the bytes to the hash-generator.
4. Produce a digest of the hash-generator.
5. Decode the newly generated digest into its string value as specified in Appendix D.

* “hash-generator” as used above indicates an in-memory instance of a hash creation object within the context of a running computer program. For example, in python3 the code “hashlib.md5()” produces a new md5 hash-generator.