ASC Media Hash List (ASC MHL)

Advanced Data Management Subcommittee

THE AMERICAN SOCIETY OF CINEMATOGRAPHERS

Implementation Guidelines

v1.0, 29. March, 2023

1. Introduction

This document serves as supplemental to the ASC MHL specification to assist on-set data management tool manufacturers in implementing ASC MHL in their tools as well as provide best practices for using ASC MHL to assure integrity of media content as it moves between various facilities.

Typical scenarios include camera media asset flows from on-set to various facilities such as post, VFX, and etc, as shown below:

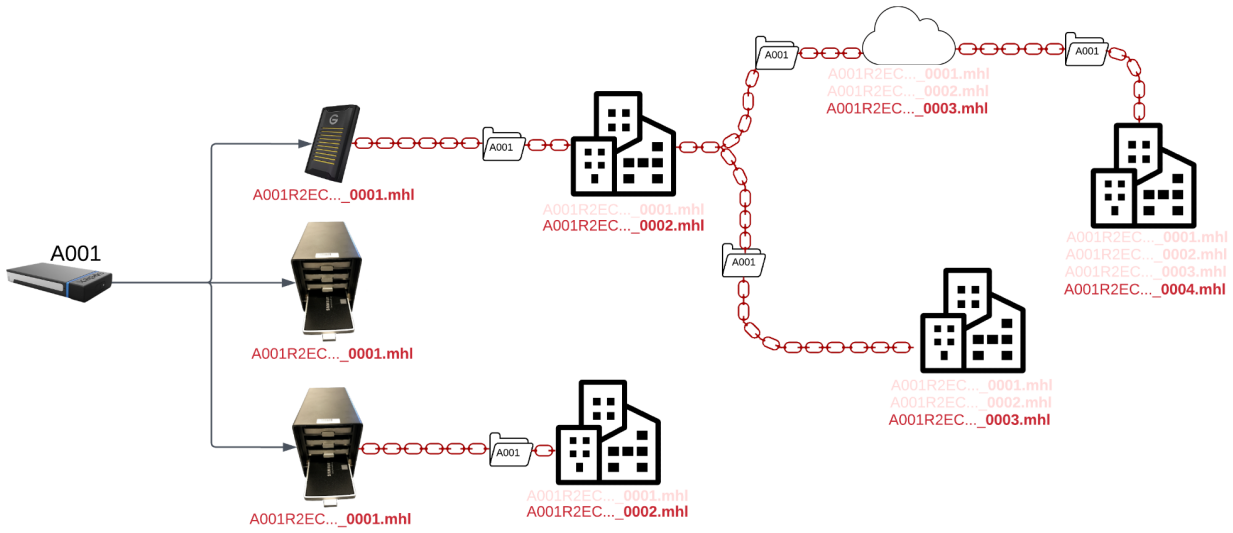

Figure 1 : Example chain of custody in movie production using ASC MHL.

2. Implementation Guidelines

2.1. Combining ASC MHL Operations for Data Management Use Cases

The ASC MHL specification defines a set of ASC MHL Operations. These operations can be combined to implement various use cases of media management activities such as starting and

extending a chain of custody as shown in Figure 1. The following sections provide a few examples of such use cases:

2.1.1. Initialize an ASC MHL History

Scenario: Offloading production files is the beginning of the life cycle of production data management. Typically, files recorded to a capture device (e.g., camera card) are copied to an external disk drive, RAID, or other backup storage volume. At this time, an ASC MHL compliant data management tool shall create an ASC MHL Manifest containing the hashes generated from all the files on the source via an ASC MHL Create operation. These hashes would have a `HashFormatType` of `original`.

Use Case: Create a new ASC MHL history for a file system folder. By adding an ASC MHL History to a file system folder it becomes a “managed data set” as defined in the ASC MHL specification.

Example: A camera card is copied to a travel drive and a new ASC MHL History is created with an ASC MHL Manifest file containing hashes for all the files on the card.

Operations: Create (section 5.6.2.)[1]

2.1.2. Verify a Managed Data Set

Scenario: An external disk drive containing an ASC MHL History for its content is received at a facility, and the contents of the drive are copied onto a facility storage volume. At this time, an ASC MHL compliant tool is used to generate hashes for the transferred files from the destination drive (e.g., file server). The tool generates a new ASC MHL Manifest using an ASC MHL Create operation with these hashes. The tool compares the generated hashes against the hashes from the ASC MHL Manifests on the source drive via the ASC MHL Verify function. If they match, the tool labels these hashes as `verified` in the `HashFormatType` element of the newly created ASC MHL Manifest in the destination.

Use Case: A data management operation (such as backup or restore) is performed on a managed data set. The files are checked for completeness, verified, and a new generation manifest is created in the ASC MHL History as a record of the data management activity.

Example: A copy of a camera card is made (e.g. offloading a camera card to a travel drive), and then a copy of the copy (e.g. copying the card on the travel drive to a file server). Two ASC MHL Manifests are created during the overall process, documenting the history of both copy processes. The creation of the second manifest happens as part of the verification use case.

Operations: ASC MHL Create (section 5.6.2), ASC MHL Diff (section 5.6.3), ASC MHL Verify (section 5.6.4), ASC MHL History Append (section 5.6.5)

[1] Section numbers refer to the ASC MHL specification document.

2.1.3. Add files to an existing folder with an ASC MHL History

Scenario: In this scenario, additional files (for example, set reports or audio files from an external recorder) are copied to a folder with an already existing ASC MHL History. At this time, the ASC MHL compliant data management tool generates a new ASC MHL Manifest containing hashes for the newly added files only and becomes part of the existing ASC MHL History. The files that were already present in the folder are untouched by the data management process, so neither a completeness check nor an integrity check are performed on these.

Use Case: Add records of single files to an existing ASC MHL History.

Example: Add a report PDF to a managed data set.

Operations: ASC MHL Create (section 5.6.2), ASC MHL History Append (section 5.6.5)

2.1.4. Nesting

Scenario: Copies of multiple camera cards are created as subfolders in a “shooting day” folder on an on-set storage volume using an ASC MHL compliant data management tool. During this process, the tool will create discrete ASC MHL Histories containing the first generation of hashes of all files in the individual subfolders, i.e. one ASC MHL History per camera card.

At the end of the day, the “shooting day” folder is copied in a single data management process to a travel drive. During this process, the data management tool will create multiple new generations in the ASC MHL Histories as follows:

- Each subfolder for a camera card gets a second generation ASC MHL Manifest containing verified hashes of files in the individual camera card folder. This ensures that if the camera card folder is moved by itself, it contains a complete ASC MHL History of the files.
- At the “shooting day” folder level (the root folder of the process) a first generation ASC MHL Manifest is created containing references to all added ASC MHL generations of the “nested” camera card folders. This new ASC MHL History also contains original hashes of files located outside the folders that already had an ASC MHL History. This ensures that the entire ASC MHL History of the shooting day folder contains complete information about all the files and folders in the root folder.

To be compliant with the ASC MHL specification, any future data management activity performed by the tool on such nested folder structures should recursively process all nested ASC MHL Histories stored in subfolders, starting with the ASC MHL History at the root folder.

Use Case: Grouping two or more managed data sets into a shared new root folder and recording this data management activity in the existing ASC MHL Histories as well as in a new ASC MHL History on the root level. Please see section 5.3.2 of the ASC MHL specification for an illustration of nesting.

Example: Copying two camera mags to a *Reels* folder on a travel drive, and then copying the entire *Reels* folder to a server.

Operations:

- For each folder: ASC MHL Diff (section 5.6.3), ASC MHL Verify (section 5.6.4), ASC MHL Create (section 5.6.2), ASC MHL History Append (section 5.6.5)
- On the root level: ASC MHL Create (with hash list references) (section 5.6.2)

2.1.5. Renaming

Use Case: Recording name changes of files in a managed data set.

Example: Rename camera files with a corrected camera letter (e.g. *A001C010.mxf* → *B001C010.mxf*)

Operations: ASC MHL Rename (section 5.6.6)

2.1.6. Flattening

Scenario: An ASC MHL data management tool generates a "packing list" file by creating a flattened ASC MHL Manifest from the nested ASC MHL History of a travel drive. This manifest is sent independently to the receiver of the travel drive as a “heads-up” description of the data that can be expected to be received.

Use Case: Creating one ASC MHL Manifest consisting of all or a selection of files from an ASC MHL History, potentially with nested ASC MHL Histories.

Operations: ASC MHL Flatten (section 5.6.7)

2.2. Using the “creatorinfo” and “comment” Elements

The `creatorinfo` element of an ASC MHL manifest carries information that applies to an entire manifest. It includes the creation date, information about the location and host where the manifest was created, and which software tool performed the creation. It is recommended that an ASC MHL compliant tool uses one or more `author` elements to specify information about the person(s) responsible for the data management process, such as name and role. It is also recommended that the tool specifies contact information (e.g., phone or email) to help track questions that may arise during the workflow.

It is recommended that the ASC MHL complaint tool uses the `comment` field in the `creatorinfo` element to provide any additional insights into the data management process that are otherwise not covered by the other content of the ASC MHL Manifest.

Since it is an “Informational, human readable text specifying a summary of the purpose, context, and parameters of the process” (section 6.4.2), it is up to the discretion of the tool manufacturer to either add meaningful information automatically or prompt the user to choose such information.

Below are some examples of information that the `comment` field could carry:

- Details about absolute paths.
    - E.g. *Backup of /Volumes/InternalRAID1/Cards/2022-07-18/A030 to /Volumes/TravelDrive10.*
    - This is useful information since the ASC MHL Manifest includes only relative paths. Here `comment` is being used to carry information on the type of data management process (creating a new backup of the media), the absolute source path (a folder A030 on volume InternalRAID1), and the absolute destination (on volume TravelDrive10).
- Carry Information about verified media.
    - E.g. *Verification of /Volumes/TravelDrive10/A030.*
- Hold that information about additional copies, which can help identify additional "branches" of backup histories.
    - E.g. *Backup of /Volumes/InternalRAID1/Cards/2022-07-18/A030 to /Volumes/TravelDrive10 (simultaneous backup to /Volumes/TravelDrive11).*

The above few examples are just to showcase the potential of the `comment` field to carry important information and is meant to be used as a starting point for tool implementers. It can of course be modified or extended with additional information. Implementations should not expect the `comment` field to conform to a specific format or wording as it is free text.

2.3. Using Multiple Hash Formats

An ASC MHL complaint tool can choose to use a hash algorithm different from the one used to create the `original` hash when creating a backup to the file server (i.e., the second generation). There might be several reasons for doing so, such as to ensure compatibility with other data management systems.

For example, let's say the hashes for the first generation, when offloading from a camera card, are created using the `xxh64` hash format by the DIT. The second copy in the post facility is verified against the existing `xxh64` hashes, and additional `C4` hashes are created and stored. During this process both hash formats are created for all the files, `xxh64` for verification, and `C4` for the new ASC MHL Manifest.

Therefore, the ASC MHL Manifest for the second generation will now contain two hash values for each hash record, both labeled as `verified`.

When calculating multiple hash algorithms on a given file, it is recommended that hash calculations be performed from a single file read operation to ensure that all hashes can be used later for verification with the same level of trust.

If the same file is read multiple times, (e.g., once per algorithm), it is possible (however unlikely) for the file contents to have changed between reads. If the file does indeed change between reads, labeling the latter hash in the ASC MHL Manifest as `verified` becomes a false negative; in this

case the only way a user could identify the issue is if an ASC MHL Verify operation is performed on the file using the original algorithm.

2.4. Ignoring files

The ASC MHL specification allows for indicating certain files to be ignored during verification. The “ignore” feature is used when files exist in the managed data set, but the data management tool is not supposed to trigger errors during verification stemming from changes in these files. Single files can be identified individually, or multiple files can be tagged for ignoring using patterns (e.g. by specifying a file extension or subfolder). Please see section “5.6.1.2. Ignore Semantic” and Appendix C in the ASC MHL specification for more details.

Example: When recording with a RED camera, media and metadata is stored in R3D files (extension .R3D) on the camera card. When the user opens an R3D file with the REDCINE-X application and edits metadata (such as RAW settings), the new clip metadata is stored in an additional RMD file (extension .RMD) with the same basename as the opened R3D file. This new file is not considered to be part of the OCF so it can be tagged for ignoring in the verification process.

During the lifetime of an ASC MHL History the list of ignored files can only grow. This means that files cannot be “un-ignored” without creating a new history.

2.5. Directory Hashes

The directory hashes specified in the ASC MHL Specification offer an additional way (besides comparing individual file hashes) of verifying the integrity of directory structures. A directory hash consists of a pair of hash values, the “content hash value” and the “structure hash value” (please see section 6.5.2 DirectoryHashType in the specification document for details).

2.5.1. Content Hash Value

The content hash value represents the content of the files and subdirectories at a given directory level. Below are some guidelines for tool manufacturers for implementing content hash value:

- Changing the content of a file in the directory changes the content hash value.
- Changing the name of a file or subfolder at that directory level does *not* change the content hash value.
- Moving a file to another folder in the directory hierarchy changes the content hash value.
- Changing content hashes in subfolders changes the content hash value.

2.5.2. Structure Hash Value

The structure hash value represents the content and naming of the files and subdirectories at a given directory level. Below are some guidelines for tool manufacturers for implementing content hash value:

- Changing the content of a file in the directory changes the structure hash value.
- Changing the name of a file or subfolder at that directory level changes the structure hash value.
- Moving a file to another folder in the directory hierarchy changes the structure hash value.
- Changing structure hashes in subfolders changes the structure hash value.

2.5.3. Creating Directory Hashes

An ASC MHL compliant tool will create directory hashes in a recursive process that uses the file hashes of all files in a directory and its subdirectories as well as hashes of the names of files and directories to create a hash pair for each directory on every level of a file hierarchy.

In general, directory hashes shall be calculated and included in the manifests, but there can be situations where directory hashes cannot be created during an ASC MHL Operation. One such scenario is described below:

Normally, an ASC MHL Manifest contains hashes that have been calculated from files during the current ASC MHL Operation. This means that directory hashes can only be recorded in a manifest if hashes are created for all files in a directory and all of its subdirectories. When a new generation of hashes is created just for a few newly added files to a larger existing data set, it would be very time consuming to hash the entire data set in order to update the directory hashes. In such cases directory hashes can be omitted in the new ASC MHL History.

If directory hashes are required at a later time, it is possible to create a new generation by verifying the entire folder structure including the creation of directory hashes at any time.

2.5.4. Using Directory Hashes as Receipts

When using directory hashes, each directory in the folder structure of a managed data set is assigned a set of directory hashes. As these directory hashes can be easily checked manually, the directory hash of the root directory of the managed data set can be used for manually checking consistency and completeness of an entire folder structure against another copy.

Example: After copying the content of the travel drive to the file server, the data management operator at the facility manually reads the directory hash of the root level of the ASC MHL History from the latest ASC MHL directory and sends it in an email as a “receipt” to the data management

operator on set. Then the data management operator on set will compare the hash that was received with the root-level directory hash in the ASC MHL Manifest of the first generation.

This process of hash exchange between the facility and on-set data management operator offers an additional, manually checkable layer of confirmation that the same file and folder content has reached the file server in the facility and that it is safe to delete the camera cards on set.

2.6. Flattening Histories and Collection Files

Transferring large amounts of data to another location, possibly even in multiple batches or volumes, requires extra care to ensure completeness of the entire data transfer. While ASC MHL Histories can ensure completeness and integrity of individual folder structures and volumes, the absence of one such folder structure or volume can obviously not be detected with the help of the manifest files within that folder structure.

In order to solve this issue, an ASC MHL specification introduced the concept of flattening. An ASC MHL History (consisting of multiple manifests) can be “flattened” into a new, independent, single manifest file containing all information to check integrity of the managed data set of the history. Such manifest files are called “packing lists.” Multiple of these packing lists can be grouped in one folder. An ASC MHL Collection file (section 5.5 of the specification) lists all the packing lists and their file hashes. A collection of packing list manifests and the collection file can be sent (for example, as a ZIP file) to the receiving side of a data transfer independently of the transferred data, or through a different mode of communication, as a heads-up. The receiver can check if all expected folder structures or volumes are received using this ASC MHL Collection.

Packing list manifests contain relative paths and hash values of all files in the managed data set and thus can be used directly to verify the integrity of the received data set. This is a good way to detect missing or lost ASC MHL History data.

2.7. Handling Warnings and Errors

The operations defined in the ASC MHL Specification (for example, the “ASC MHL Verify” or “ASC MHL Create” operation) can execute complex calculations and long-running processes. During these processes a range of errors and warnings can occur – from standard file issues such as file access errors, to ASC MHL-specific cases such as hash mismatches. It is important for users to have commonly occurring errors presented to users in a standard way and at specific times across a multitude of ASC MHL compliant tools. Listed below are some commonly encountered error scenarios that all ASC MHL complaint tools should handle and display clear messages about to their users:

- A hash mismatch with a file of the Managed Data Set is detected: A hash created from a file of the Managed Data Set doesn't match a hash already recorded in the ASC MHL History for the same file. (see below 2.7.1 section)
- A missing file in the Managed Data Set is detected: A file is recorded in the ASC MHL History but not found in the Managed Data Set.
- A hash mismatch with an ASC MHL manifest file is detected: A hash created from a ASC MHL Manifest file doesn't match the hash in the ASC MHL Chain or ASC MHL Collection file.
- A missing ASC MHL manifest file is detected: A file is listed in an ASC MHL Chain or ASC MHL Collection file but not found at the expected file location.
- An additional file is found in the Managed Data Set: When executing an ASC MHL Operation on an entire directory, a list of files that have not yet been recorded in the ASC MHL History should be presented to the user. These files are also added to the new generation.
- When performing an ASC MHL Verify operation against a read-only volume, an ASC MHL complaint tool should show a message saying the new generation cannot be written due to read-only permissions.

It is recommended that an ASC MHL complaint tool follow the guidance listed below on presentation and behavior of warnings and errors:

- Warnings and errors that occur during the execution of a command shall be aggregated and displayed to the user as soon as they occur, if possible. When various unrelated errors occur, for example, within nested histories, just showing the first or last error might give a misleading image of the underlying problem.
- The occurrence of errors and warnings during the execution of a command shall be displayed to the user prominently.
- Generally errors and warnings should not abort an ASC MHL Operation or data management process.

2.7.1 Handling Hash Mismatches During Verification

When a new hash mismatch is detected during verification, it is recorded in the resulting ASC MHL Manifest by setting the `action` attribute of the offending file's `HashFormatType` to failed (section 6.5.1). As an option, the tool performing the verification may recalculate the hash to confirm the mismatch before recording it in the manifest.

When using existing ASC MHL Manifests for verification, any hash labeled as `failed` should not be used. This ensures that files are only ever verified against their originally-calculated hashes, which will be labeled as `original` or `verified`.

Additionally, hashes labeled as `failed` should not be used during any ASC MHL Flatten operations (section 5.6.7).

2.8. Recommendations for Renaming Files in Managed Data Sets

In case a data management workflow includes renaming files, it is recommended that the data management software covers that renaming in order to track the file name changes in the ASC MHL History. Renaming files without records in the ASC MHL History would lead to failed verification, and the renamed file would likely be registered as a new file with a new hash during the next data management activity.

It is up to the discretion of the manufacturer to automatically find renamed files (for example by matching file hashes) and add new and previous file names to the ASC MHL Manifest.

2.9. Supporting Previous MHL

Although ASC MHL is conceptually similar to the original Media Hash List (MHL) specification, and there are certain commonalities, ASC MHL is not a derivative of MHL and is not backward compatible with it. Therefore, it is recommended that the tools continue to support the original MHL, if it is already implemented. However, ASC MHL can be considered a successor of MHL and thus be the default option in cases where both formats are supported.

3. Implementing Support for ASC MHL Using Reference Implementation

The reference implementation at https://github.com/ascmitc/mhl offers a rich set of features targeting all aspects of the ASC MHL specification. Aspects such as multiple hash formats, nesting, and ignore patterns are covered so that complex combinations of these features are handled properly.

If tool manufacturers want to base their support for ASC MHL on this implementation, the `mhllib` part of the reference implementation can be used to build custom commands (e.g. to implement writing ASC MHL manifests with hash values calculated in an external data management process as one activity).

The Python implementation can, for example, be packaged to be embedded in other applications via PyInstaller.