ascmhl

American Society of Cinematographers (ASC)

Sep 06, 2022

CONTENTS:

This documentation is for the ASC MHL command line tool `ascmhl` hosted on https://github.com/ascmitc/mhl.

The ASC MHL tool can

- create and extend ASC MHL history for given files and entire file hierarchies in a file system,
- output information about recorded history, and
- verify files and entire file hierarchies.

The "commands" section of this documentation is auto-generated from the Python source code, while the "ASC Media Hash List (ASC MHL)" section with detailled information is imported from the REAMDE.md of the ghithub repoistory at https://github.com/ascmitc/mhl.

CHAPTER

ONE

ASC MEDIA HASH LIST (ASC MHL)

> The software in this repository is the reference implementation for the ASC Media Hash List (ASC MHL) format specification by the Advanced Data Management Subcommittee of the ASC Motion Imaging Technology Council (MITC) at the American Society of Cinematographers (ASC).
>
> Resources:
>
> - ASC MHL Specification page (at theasc.com)
>
> In case you are looking for the original specification of MHL, please take a look at https://mediahashlist.org.

Ensuring file integrity when backing up and verifying files during production and post-production is of utmost importance. The ASC MHL is used to create a chain of custody by tracking each and every copy made between the media's initial download on set, all the way through to final archival.

The ASC MHL uses common checksum methods for hashing files and folders, specifies what information is gathered, where the checksum is placed, and documents these hashes together with essential file metadata in an XML format that is human-readable.

This repository holds all information about the document format, a reference implementation, and tools.

1.1 ASC MHL Format Specification

The ASC MHL consists of a

- definition of naming conventions for the ascmhl folder and the file names of its content
- XML schema for the ASC MHL files
- definition for the chain file

The schema definition can be found in the `./xsd` folder.

1.2 `mhllib` Reference Implementation

The implementation of a reference library aims to be used in applications and tools dealing with ASC MHL files. The library takes responsibility of dealing with complex use cases of nesting and assembling of information.

The reference library covers

- reading ascmhl folders and their contents
- parsing and writing of ASC MHL XML files
- parsing and writing ASC MHL chain and collection files

- dealing with nested mhl folders

ASC MHL supports the hash formats

- xxHash (64-bit, and latest XXH3 with 64-bit and 128-bit)
- MD5
- SHA1, SHA256
- C4

The source code for `mhllib` can be found in the `./ascmhl` folder.

1.3 The `ascmhl` Tool

The `ascmhl` tool is a command line tool based on `mhllib` that allows to perform typical activities for the use cases of ASC MHL.

- *The `create`command*: Create a new generation for a folder or file(s)
- *The `diff`command*: Diff an entire folder structure
- *The `flatten`command*: Flatten an MHL history into one external manifest
- *The `info`command*: Prints information from the ASC MHL history

Typical scenarios, sample CLI output, and generated ASC MHL files can be found in the README.md file in the `examples/scenarios` folder of the git repository.

The documentation can also be found at https://ascmhl.readthedocs.io/

1.4 The `ascmhl-debug` Tool

The `ascmhl-debug` tool is a command line tool with additional operations and commands that might come in handy during implementation or testing.

- *The `verify`command*: Verify a folder, single file(s), or a directory hash (without writing a new generation)
- *The `xsd-schema-check`command*: Checks a .mhl file against the xsd schema definition
- *The `hash`command*: Create and print a hash value for a file

1.5 Getting started

The `mhllib` as well as the `ascmhl` and `ascmhl-degug` tools require a few dependencies that need to be installed first.

For installing system dependencies on macOS Homebrew is recommended.

1.5.1 System requirements

Make sure you have Python 3 installed:

```
$ brew install python3
$ brew postinstall python3
```

1.5.2 Installing / updating `ascmhl` as a user

Please run the following command to install (or upgrade to) the latest development version of `ascmhl`:

```
$ pip3 install --upgrade ascmhl
```

To verify that it has been correctly installed run:

```
$ ascmhl --help
```

1.5.3 Installing `ascmhl` as a developer

Download the source code and install dependencies using a Virtual Environment:

```
$ git clone https://github.com/ascmitc/mhl.git
$ cd mhl
$ python3 -m venv env
$ source env/bin/activate
$ pip3 install --editable .
```

This will install the wrapper scripts `ascmhl` and `ascmhl-debug` to be available on your `$PATH`. Inside the virtualenv, this wrapper will be installed as `env/bin/ascmhl`. Regular users might have it in `/Library/Frameworks/Python.framework/Versions/3.9/bin/ascmhl` or `/usr/local/bin`. For Windows users, pip will create an `ascmhl.exe` and an `ascmhl-debug.exe`.

More information on installing Python commandline tools using `entry_points` can be found here:

- https://setuptools.readthedocs.io/en/latest/userguide/entry_point.html
- https://packaging.python.org/specifications/entry-points/#use-for-scripts
- https://click.palletsprojects.com/en/master/setuptools/

Adding the `-e / --editable` flag installs a linked version to your `site-packages` directory to allow editing the source files in your working directory as usual.

1.6 Common Scenarios for `ascmhl`

The `ascmhl` tool can be used to

- verify and create new MHL generations for given files and folders (command `create`),
- print differences between the records in the MHL history and given files and folders (command `diff`),
- create one "flattened" manifest file from a history (command `flatten`), and
- print information about an MHL history (command `info`).

1.6.1 Working with file hierarchies (with completeness check)

The most common commands when using the `ascmhl` in data management scenarios are the `create` and the `check` commands in their default behavior (without subcommand options).

Creating a new generation for a folder / drive with the `create` command traverses through a folder hierarchy, hashes all found files and compares the hashes against the records in the `ascmhl` folder (if present). The command creates a new generation (or an initial one) for the content of an entire folder at the given folder level. It can be used to document all files in a folder or drive with all verified or newly created file hashes of the moment the `create` command ran.

The `diff` command also traverses through the content of a folder / drive. The `diff` command thus behaves like the `verify` command, but the `diff` command does not hash any files (e.g. doesn't do file verification) and thus is much faster in execution. It can be used to print all files that are existent in the file system and are not registered in the `ascmhl` folder yet, and all files that are registered in the `ascmhl` folder but that are missing in the file system.

Checking a folder / drive with the `verify` command (of the `ascmhl-debug` tool) traverses through the content of a folder, hashes all found files and compares the hashes against the records in the `ascmhl` folder. The `verify` command behaves like a `create` command (both without additional options), but doesn't write new generations. It can be used to verify the content of a received drive with existing ascmhl information.

1.6.2 Working with single files (without completeness check)

In some scenarios working with an entire folder structure is not adequate, and finer control of the processes files is needed. For those scenarios the `create` command is used with additional subcommand options.

Adding single files in a new generation with the `create -sf` ("single files, no completeness check") command allows to add single files to an existing folder structure and create new generations only with records of these files.

The `info -sf` ("single file") command prints the known history of a single file with details about all generations.

Hashing and verifying single files against hash information stored in the `ascmhl` folder with the `verify -sf` ("single files") command (of the `ascmhl-debug` tool) allows to "check" single files without the need for a (probably much longer running) check of the integrity of the entire folder structure.

1.7 Commands of `ascmhl`

Implementation status 2022-09:

- **Implemented**: `create`, `flatten` (partially), `diff`, `info` (partially)

1.7.1 The `create` command

The `create` command hashes all files given with the different options and creates a new generation in the mhl-history with records for all hashed files. The command compares the hashes against the hashes stored in previous generations if available.

`create` default behavior (for file hierarchy, with completeness check)

The `create` command traverses through a folder hierarchy (such as a folder with media files, a camera card, or an entire drive). The command hashes all files (not ignored by the given ignore patterns given with the `-i` or `-ii` options) and the hashes are compared against records in the `ascmhl` folder. It records all hashed files in the new generation. Directory hashes are computed and also recorded in the new generation.

The command detects, prints error, and exits with a non-0 exit code if it finds files that are registered in the `ascmhl` folder but that are missing in the file system.

Files that are existent in the file system but are not registered in the `ascmhl` folder yet, are registered as new entries in the newly created generation(s).

The `create` command takes the root path of the file hierarchy as the parameter:

```
$ ascmhl create [-i ignore pattern|-ii /path/to/ignore-file.txt] [creator-info␣
↪options] /path/to/folder/
```

Creator-info options:

- `--location`: Location value of the `<creatorinfo>`element.
- `--comment`: Comment value of the `<creatorinfo>`element.
- `--author_name`: Name value of the `<author>` element in the `<creatorinfo>`element.
- `--author_email`: Email value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).
- `--author_phone`: Phone value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).
- `--author_role`: Role value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).

It works on folders with or without an `ascmhl` folder within the given folder hierarchy, and creates a new `ascmhl` folder at the given folder level if none is present before.

`ascmhl` folders further down the file hierarchy are read, handled, and referenced in top-level `ascmhl` folders. Existing `ascmhl` folders further down the folder structure will also get a new generation added.

Implementation:

```
read (recursive) mhl history (mhllib)
traverse folder
         hash each file
        if `ascmhl` folder exists, compare hash (mhllib)
        on error (including mismatching hashes):
                print error
                 continue
         add files to new generation if not present yet
compare found files in file system with records in ascmhl folder and \
   warn if files are missing that are recorded in the ascmhl folder
create new generation(s) (mhllib)
```

`create` with `-sf` option(s) (for single file(s), no completeness check)

The `create` command with `-sf` option is run with the root path of the file hierarchy as well as one or multiple paths to the individual files to be recorded as the parameters.

This command can be used for instance when adding single files to an already mhl-managed file hierarchy.

```
$ ascmhl create /path/to/root/folder -sf /path/to/single/file1 [-sf /path/to/single/
↪file2 ..]
```

A new generation is created in all `ascmhl` folders below the given root path (e.g. in a nested mhl-history). If no mhl-history is present yet, an error is thrown.

No other files than the ones specified as `-sf` options are handled by this command.

Implementation:

```
read (recursive) mhl-history (mhllib) starting from root path
for each file from input
        check if file is not recorded in `ascmhl` folder yet
        hash file
        add record for file to new generation (mhllib)
                add a new generation if necessary in appropriate `ascmhl` folder␣
↪(mhllib)
```

1.7.2 The `flatten` command

The `flatten` command takes the root path of the file hierarchy and the destination path for the flattened manifest as the parameter:

```
$ ascmhl flatten [-i ignore pattern|-ii /path/to/ignore-file.txt] [creator-info␣
↪options] /path/to/folder/ /destination/path/
```

Creator-info options:

- `--location`: Location value of the `<creatorinfo>`element.
- `--comment`: Comment value of the `<creatorinfo>`element.
- `--author_name`: Name value of the `<author>` element in the `<creatorinfo>`element.
- `--author_email`: Email value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).
- `--author_phone`: Phone value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).
- `--author_role`: Role value of the `<author>` element in the `<creatorinfo>`element (`--author_name` must also be set for this option).

TBD

```
% ascmhl flatten --help
Usage: ascmhl flatten [OPTIONS] ROOT_PATH DESTINATION_PATH

  Flatten an MHL history into one external manifest

  The flatten command iterates through the mhl-history, collects all known files and
  their hashes in multiple hash formats and writes them to a new mhl file outside of␣
↪the
```

(continues on next page)

(continued from previous page)

```
  iterated history.

Options:
  -v, --verbose                Verbose output
  -n, --no_directory_hashes    Skip creation of directory hashes, only reference
                               directories without hash
  -i, --ignore TEXT            A single file pattern to ignore.
  -ii, --ignore_spec PATH      A file containing multiple file patterns to
                               ignore.
  --author_name TEXT           Name value for the <author> element in the
                               <creatorinfo> element
  --author_email TEXT          Email value for the <author> element in the
                               <creatorinfo> element
  --author_phone TEXT          Phone value for the <author> element in the
                               <creatorinfo> element
  --author_role TEXT           Role value for the <author> element in the
                               <creatorinfo> element
  --location TEXT              Value for the <location> element in the
                               <creatorinfo> element
  --comment TEXT               Value for the <comment> element in the
                               <creatorinfo> element
  --help                       Show this message and exit.
```

1.7.3 The `diff` command

The `diff` command is very similar to the `verify` command in the default behavior, only that it doesn't create hashes and doesn't verify them. It can be used to quickly check if a folder structure has new files that have not been recorded yet, or if files are missing.

The command detects, prints errors, and exits with a non-0 exit code for

- all files that existent in the file system but not registered in the `ascmhl` folder yet, and
- all files that are registered in the `ascmhl` folder but that are missing in the file system.

It is run with the root path of the file hierarchy as the parameter.

```
$ ascmhl diff /path/to/folder/
```

If no `ascmhl` folder is found on the root level, an error is thrown.

`ascmhl` folders are read recursively.

Implementation:

```
error if no mhl folder found on root level
read (recursive) mhl history (mhllib)
traverse folder
        on missing file:
                print error
                 continue
compare found files in file system with records in ascmhl folder \
  and warn if files are missing that are recorded in the ascmhl folder
end with exit !=0 if at least one of the files has failed, a file was \
  missing, or new files have been found
```

1.7.4 The `info` command

`info` default behavior

The `ascmhl` folder contains well readable XML files, but the number of recorded files, generations, hash entries, verification info and so forth adds up to an amount of information that cannot be quickly understood. The `info` command helps to get a quick overview of the contents of the stored information in an `ascmhl` folder.

The `info` command prints

- a list of generations (with the `-v` option also with creator info and process info)
- *[not implemented yet]* a summary (with the `-s` subcommand option) of the information in an ascmhl folder, such as
- number of recorded files, and a list of the generations with their creator info, and/or
- *[not implemented yet]* a list (with the `-l` option) of all file (and folder) records stored in an ascmhl folder,
- together with relative file paths, file size, and known file hashes.

It is run with the path to a specific `ascmhl` folder as the parameter.

```
$ ascmhl info [-s|-l] [-v] /path/to/ascmhl/
```

Implementation:

```
error if no mhl folder found on root level
read (recursive) mhl history (mhllib)
if summary option:
        print summary
if list option:
        for each file record
                print file info, hashes, etc.
```

`info` with the `-sf` subcommand option

The `info` command with the `-sf` subcommand option outputs information about the full and detailed history information about one file.

```
$ ascmhl info -sf /path/to/file [-sf /path/to/other/file] [/root/path]
```

The command outputs each generation where the file has been handled, including date, hash, and activity (and creator info and absolute path with the `-v` option). The history information is read from the "next" ASC MHL history found in the path, of at the given root path.

Implementation:

```
find mhl-history information in the path above (mhllib)
        error of no `ascmhl` folder is found
print detailed info for file
```

`info` with the `-dh` subcommand option *[not implemented yet]*

The `info` command with the `-dh` subcommand option prints

- the directory hash of a folder computed from stored file hashes of an `ascmhl` folder (with the `-dh` option).

The directory hash can be used to quickly verify if the state of a folder structure is still the same compared to the last generation created with a `create` command (manually compare with the hash in the `<root>` tag in the ASC MHL file).

It is run with the path to a specific `ascmhl`folder and the path to the desired folder for the computed directory hash.

```
$ ascmhl info -dh /path/to/ascmhl/ /path/to/sub/folder
```

Implementation:

```
error if no mhl folder found on root level
read (recursive) mhl history (mhllib)
calculate directory hash from file hashes
print directory hash
```

1.8 Commands of `ascmhl-debug`

1.8.1 The `verify` command

`verify` default behavior (for file hierarchy, with completeness check)

The `verify` command traverses through the content of a folder, hashes all found files (filtered by the ignore patterns from the `ascmhl` folder) and compares the hashes against the records in the `ascmhl` folder.

The command detects, prints errors, and exits with a non-0 exit code for

- all files that are existent in the file system but are not registered in the `ascmhl` folder yet, and
- all files that are registered in the `ascmhl` folder but that are missing in the file system.

It is run with the root path of the file hierarchy as the parameter.

```
$ ascmhl verify /path/to/folder/
```

If no `ascmhl` folder is found on the root level, an error is thrown.

`ascmhl` folders further down the file hierarchy are also read, and its recorded hashes are used for verification.

Implementation:

```
error if no mhl folder found on root level
read (recursive) mhl history (mhllib)
traverse folder
        hash each file (filtered by ignore patterns from mhl folder)
        compare hash (mhllib)
        on error (including mismatching hashes):
                print error
                 continue
compare found files in file system with records in ascmhl folder and \
  warn if files are missing that are recorded in the ascmhl folder
end with exit !=0 if at least one of the files has failed, a file was \
  missing, or new files have been found
```

`verify` with `-sf` option (for single file, no completeness check)

The `verify` command can be used to verify a single file. It is run with the path to a single file as the parameter.

The path can be

- the relative path to the file starting from the root folder of the history, or
- the absolute path to the file.

```
$ ascmhl verify -sf /absolute/path/to/single/file
$ ascmhl verify -sf realtive/path/to/single/file
```

The command looks for an `ascmhl` folder in the folders above the given files. If no mhl-history is present yet, an error is thrown.

Implementation:

```
find mhl-history information in the path above (mhllib)
        error of no `ascmhl` folder is found
read (recursive) mhl-history (mhllib)
for file from input
        hash file
        compare hash (mhllib)
if file is not found in mhl-history, throw error
on error (including mismatching hashes):
        don't break
        print error
        end with exit !=0 if the verification has failed
```

`verify` with `-dh` subcommand option (for directory hash)

The `verify` command with the `-dh` subcommand (or `--directory_hash`) option creates the directory hash by hashing the contained files of the given directory path (filtered by the ignore patterns from the `ascmhl` folder) and compares it with the to-be-expected directory hash calculated from the file hashes (same calculation as the `info` command with the `-dh` subcommand option).

```
$ ascmhl verify -dh [-co [-ro]] /path/to/folder
```

The `-co` option (or `--calculate_only`) only calculates and prints the directory hashes and doesn't verify them against an existing history. This option also works when no history is present. The `-ro` option (or `--root_only`) only calculates and prints the root directory hash. This option is only in effect with the `-co` option.

Implementation:

```
find mhl-history information in the path above (mhllib)
        error of no `ascmhl` folder is found
read (recursive) mhl history (mhllib)
calculate to-be-expected directory hash from file hashes
traverse folder
         hash each file
calculate actual directory hash
compare to-be-expected directory hash with actual directory hash
on error (including mismatching hash):
        print error
        end with exit !=0
```

`verify` with `-pl` subcommand option (for packing lists)

The `verify` command with the `-pl` subcommand (or `--packing_list`) option can be used to verify a folder structure with a given packing list.

It is run with the path to the packing list manifest file as the parameter.

```
$ ascmhl verify -pl /path/to/packing-list.mhl
```

TBD

1.8.2 The `xsd-schema-check` command

The `xsd-schema-check` command validates a given ASC MHL Manifest file against the XML XSD. This command can be used to ensure the creation of syntactically valid ASC MHL files, for example during implementation of tools creating ASC MHL files.

Note: The `xsd-schema-check` command must be run from a directory with a `xsd` subfolder where the ASC MHL xsd files are located (for example it can be run from the root folder of the ASC MHL git repository). Alternatively you can pass the local path to the XSD file (available here) with the `-xsd` or `--xsd_file` option.

```
$ ascmhl xsd-schema-check /path/to/ascmhl/XXXXX.mhl
```

`xsd-schema-check` with the `-df` subcommand option

The `xsd-schema-check` command with the `-df` subcommand option can validates a ASC MHL Directory file instead of a manifest file.

It is run with the path to a ASC MHL Directory file.

```
$ ascmhl xsd-schema-check -df /path/to/ascmhl/ascmhl_chain.xml
```

1.8.3 The `hash` command

The `hash` command hashes an individual file with the given hash algorithm (via `-h` or `--hash_format`) and prints the hash value.

```
$ ascmhl-debug hash --help
Usage: ascmhl-debug hash [OPTIONS] FILE_PATH

  Create and print a hash value for a file

Options:
  -h, --hash_format [md5|sha1|xxh128|xxh3|xxh64|c4]
                                  Algorithm  [required]
  --help                          Show this message and exit.
```

1.9 Known issues

The current state of the implementation is intended to give a good overview what can be done with ASC MHL. Nonetheless this is not yet a complete implementation of the ASC MHL specification:

- Currently not all initially specified commands are implemented yet (see sections above)
- Renaming of files is currently not implemented (neither as command, nor proper handling in histories and packing
- lists)
- The chain file is currently not verified yet

Also see the GitHub issues page for more.

CHAPTER

TWO

COMMANDS

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

This documentation is created automatically from https://github.com/ascmitc/mhl.