cdl_convert Documentation

Release 0.9.2

Sean Wallitsch

April 10, 2016

Contents

1 Project Info **1**

2 Introduction **3**

3 Changelog **5**

4 Table of Contents **9**

4.1 Script Usage . . . 9
4.2 ColorCorrection Usage . . . 10
4.3 Color Collections . . . 15
4.4 Installation . . . 20
4.5 Changelog . . . 20
4.6 Support . . . 24
4.7 Frequently Asked Questions . . . 25
4.8 Contributing . . . 25
4.9 Licenses . . . 27
4.10 API Reference . . . 29

5 Indices and tables **49**

CHAPTER 1

Project Info

- **Name:** cdl_convert
- **Version:** 0.9.2
- **Author/Maintainer:** Sean Wallitsch
- **Email:** shidarin@alphamatte.com
- **License:** MIT
- **Status:** Development
- **Docs:** http://cdl-convert.readthedocs.org/
- **GitHub:** https://github.com/shidarin/cdl_convert
- **PyPI:** https://pypi.python.org/pypi/cdl_convert
- **Python Versions:** 2.6-3.4, PyPy

CHAPTER 2

Introduction

`cdl_convert` converts between common ASC CDL formats. The American Society of Cinematographers Color Decision List (ASC CDL, or CDL for short) is a schema to simplify the process of interchanging color data between various programs and facilities.

The ASC has defined schemas for including the 10 basic numbers in 5 different formats:

- Avid Log Exchange (ALE)
- Film Log EDL Exchange (FLEx)
- CMX EDL
- XML Color Correction (cc)
- XML Color Correction Collection (ccc)
- XML Color Decision List (cdl)

Unofficial Formats:

- OCIOCDLTransform, a Foundry Nuke node
- Space separated CDL, a Rhythm & Hues internal cdl format

It is the purpose of `cdl_convert` to convert ASC CDL information between these basic formats to further facilitate the ease of exchange of color data within the Film and TV industries.

`cdl_convert` supports parsing ALE, FLEx, CC, CCC, CDL and RCDL. We can write out CC, CCC, CDL and RCDL.

cdl_convert is not associated with the American Society of Cinematographers

CHAPTER 3

Changelog

New in version 0.9.2:

- Fixed a bug where ALE's with blank lines would not convert correctly.
- Fixed a bug that was preventing `cdl_convert` from being correctly installed in Python 2.6
- Fixed continuous integration testing.
- No longer officially supporting Python 3.2, as I've had to remove it from our CI builds. It should still work just fine though, but we won't be running CI against it.

New in version 0.9:

- Added ability to parse CMX EDLs
- Fixed a script bug where a collection format containing color decisions will not have those color decisions exported as individual color corrections.
- Fixed a bug where we weren't reading line endings correctly in certain situations.
- Added a cdl_convert.py stub file to the package root level, which will allow running of the cdl_convert script without installation. Due to relative imports in the python code, it was no longer possible to call cdl_convert/cdl_convert.py directly.
- The script, when run directly from cdl_convert.py, will now write errors to stderror correctly, and exit with a status of 1.

New in version 0.8:

- Added *–single* flag. When provided with an output collection format, each color correction in the input will be exported to it's own collection.
- Giving a `ColorCorrection` a non-duplicate ID now works unless the `--halt` flag is given. This means that incoming collections that contain duplicate IDs will not fail out.

New in version 0.7.1:

- Fixed bug where ALE's without 'Scan Filename' fields could not parse correctly.

New in version 0.7:

The biggest change in 0.7 is the addition of collection format support. `.ccc`, Color Correction Collections, can now be parsed and written. `.cdl`, Color Decision Lists, can now be parsed and written. `.ale` and `.flex` files now return a collection.

- **New script flags:**
  - Adds `--check` flag to script, which checks all parsed `ColorCorrects` for sane values, and prints warnings to shell

  - Adds `-d`, `--destination` flag to the script, which allows user to specify the output directory converted files will be written to.
  - Adds `--no-ouput` flag to the script, which goes through the entire conversion process but doesn't actually write anything to disk. Useful for troubleshooting, especially when combined with `--check`
  - Adds `--halt` flag to the script, which halts on errors that can be resolved safely (such as negative slope or power values)
- Renames `ColorCollectionBase` to `ColorCollection` , since it will be used directly by both `ccc` and `cdl`.
- Adds `parse_ccc` which returns a `ColorCollection` .
- Adds `write_ccc` which writes a `ColorCollection` as a `ccc` file.
- Adds `parse_cdl` which returns a `ColorCollection` .
- Adds `write_cdl` which returns a `ColorCollection` as a `cdl` file.
- `ColorCollection` is now a fully functional container class, with many attributes and methods.
- Added `ColorDecision` , which stores either a `ColorCorrection` or `ColorCorrectionRef` , and an optional `MediaRef`
- Added `ColorCorrectionRef` , which stores a reference to a `ColorCorrection`
- Added `parent` attribute to `ColorCorrection` .
- Calling `sop_node` or `sat_node` on a `ColorCorrection` before attempting to set a SOP or Sat power now works.
- `ColorCorrection` `cdl_file` init argument renamed to `input_file`, which is now optional and able to be set after init.
- `parse_cc` and `parse_rnh_cdl` now only yield a single `ColorCorrection` , not a single member list.
- Added dev-requirements.txt (contains `mock`)
- All `determine_dest` methods now take a second `directory` argument, which determines the output directory.
- Adds `sanity_check` function which prints values which might be unusual to stdout.
- `parse_cdl` and `write_cdl` renamed to `parse_rnh_cdl` and `write_rnh_cdl` respectively.
- **`member_reset` methods:**
  - `ColorCorrection` now has a `reset_members` method, which resets the class level member's dictionary.
  - `MediaRef` also has a `reset_members` method, as does `ColorCollection`
  - `reset_all` function calls all of the above `reset_members` methods at once.
- **Renamed `cdl_file` argument:**
  - `parse_cc` `cdl_file` arg renamed to `input_file` and now accepts a either a raw string or an `ElementTree` `Element` as `input_file`.
  - `parse_rnh_cdl` `cdl_file` arg renamed to `input_file`.
  - `parse_ale` `edl_file` arg renamed to `input_file`.
  - `parse_flex` `edl_file` arg renamed to `input_file`.
- **Python Structure Refactoring**

- Moved `HALT_ON_ERROR` into the `config` module, which should now be referenced and set by importing the entire `config` module, and referencing or setting `config.HALT_ON_ERROR`
- Script functionality remains in `cdl_convert.cdl_convert`, but everything else has been moved out.
- `AscColorSpaceBase` , `AscDescBase` , `AscXMLBase` and `ColorNodeBase` now live under `cdl_convert.base`
- `ColorCollection` now lives in `cdl_convert.collection`
- `ColorCorrection` , `SatNode` and `SopNode` now live under `cdl_convert.correction`
- `ColorDecision` , `ColorCorrectionRef` and `MediaRef` now live under `cdl_convert.decision`
- All parse functions now live under `cdl_convert.parse`
- All write functions now live under `cdl_convert.write`
- `sanity_check` now live under `cdl_convert.utils`
- `reset_all` now lives under the main module

CHAPTER 4

Table of Contents

4.1 Script Usage

Most likely you'll use `cdl_convert` as a script, instead of a python package itself. Indeed, even the name is formatted more like a script (with an underscore) than the more common all lowercase of python modules.

If you just want to convert to a `.cc` XML file, the only required argument is an input file, like so:

```
$ cdl_convert ./di_v001.flex
```

You can override the default `.cc` output, or provide multiple outputs with the `-o` flag.

```
$ cdl_convert ./di_v001.flex -o cc,cdl
```

Sometimes it might be necessary to disable cdl_convert's auto-detection of the input file format. This can be done with the `-i` flag.

```
$ cdl_convert ./ca102_x34.cdl -i rcdl
```

Note: You should not normally need to do this, but it is possible especially since there are multiple formats sharing the same file extension. In this case, `.cdl` could have indicated either a space separated cdl (an `rcdl`), or an XML cdl. `cdl_convert` does it's best to try and guess which one the file is, but if you're running into trouble, it might help to indicate to `cdl_convert` what the input file type is.

By default, converted files will be written to the './converted' directory, but a custom destination directory can easily be specified with the `-d` flag.

```
$ cdl_convert ./hk416_210.ccc -d /hello_kitty/luts/cdls/
```

Warning: It's possible to pass a '.' to the `-d` flag, causing converted files to be written to the same directory as the directory you're calling cdl_convert from, and often that ends up being the same directory as the file you're converting from. If one isn't careful, there's a possibility you could overwrite the original files.

When converting large batches of color corrections, it can be helpful to know if there's anything odd about any of them. Using the `--check` flag will cause any potentially invalid numbers to be flagged and printed to the shell.

For Slope, Power and Saturation, any values below `0.1` or above `3.0` will flag. For Offset, any values below `-1.0` or above `1.0` will flag.

```
$ cdl_convert ./di_v001.flex
The ColorCorrection "a347.x700" was given a Slope value of "0.978", which
might be incorrect.
The ColorCorrection "a400.x050" was given a Saturation value of "3.1",
which might be incorrect.
```

This is especially useful when combined with the `--no-output` flag, which will enable a dry run mode and allow you to spot odd values before running.

Full help is available using the standard `--help` command:

```
$ cdl_convert --help
usage: cdl_convert [-h] [-i INPUT] [-o OUTPUT] [-d DESTINATION] [--halt]
                   [--no-output] [--check] [--single]
                   input_file

positional arguments:
  input_file            the file to be converted

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        specify the filetype to convert from. Use when
                        CDLConvert cannot determine the filetype
                        automatically. Supported input formats are: ['flex',
                        'cc', 'ale', 'cdl', 'rcdl', 'ccc']
  -o OUTPUT, --output OUTPUT
                        specify the filetype to convert to, comma separated
                        lists are accepted. Defaults to a .cc XML. Supported
                        output formats are: ['cc', 'cdl', 'ccc', 'rcdl']
  -d DESTINATION, --destination DESTINATION
                        specify an output directory to save converted files
                        to. If not provided will default to ./converted/
  --halt                turns off exception handling default behavior. Turn
                        this on if you want the conversion process to fail and
                        not continue,rather than relying on default behavior
                        for bad values. Examples are clipping negative values
                        to 0.0 for Slope, Power and Saturation, and
                        automatically generating a new id for a ColorCorrect
                        if no or a bad id is given.
  --no-output           parses all incoming files but no files will be
                        written. Use this in conjunction with '--halt' and '--
                        check' to try and track down any oddities observed in
                        the CDLs.
  --check               checks all ColorCorrects that were parsed for odd
                        values. Odd values are any values over 3 or under 0.1
                        for Slope, Power and Saturation. For offset, any value
                        over 1 and under -1 is flagged. Note that depending on
                        the look, these still might be correct values.
  --single              only write a single color decision per file when given
                        collection formats. This means that a single input CDL
                        will export multipleCDL files, one per color decision.
```

4.2 ColorCorrection Usage

Once installed with pip, importing `cdl_convert` works like importing any other python module.

```
>>> import cdl_convert as cdl
```

4.2.1 Creating ColorCorrection

Once imported, you have two choices. You can either instantiate a new, blank cdl directly, or you can parse a file on disk.

A ColorCorrection is created with the 10 required values (RGB values for slope, offset and power, and a single value for saturation) set to their defaults.

```
>>> cc.slope
(Decimal('1.0'), Decimal('1.0'), Decimal('1.0'))
>>> cc.offset
(Decimal('0.0'), Decimal('0.0'), Decimal('0.0'))
>>> cc.power
(Decimal('1.0'), Decimal('1.0'), Decimal('1.0'))
>>> cc.sat
Decimal('1.0')
```

Note: slope, offset, power and sat are convenience properties that actually reference two child objects of ColorCorrection , a SopNode and a SatNode . Calling them via cc.power is the same as calling cc.sop_node.power.

The ColorCorrection class inherits from both the AscColorSpaceBase class, and the AscDescBase class, giving it the additional attributes of input_desc (to describe the colorspace entering the correction, viewing_desc (to describe the colorspace conversions that must occur for viewing, and what type of monitor was used), and desc (which can be an infinitely long list of shot descriptions)

Direct Creation

If you want to create a new instance of ColorCorrection, you have to provide an id, for the unique cdl identifier and an optional source filename to input_file.

```
>>> cc = cdl.ColorCorrection(id='cc1', input_file='./myfirstcdl.cc')
```

Warning: When an instance of ColorCorrection is first created, the id provided is checked against a class level dictionary variable named members to ensure that no two ColorCorrection share the same id , as this is required by the specification.
Giving duplicate id will result in a number being appended to the back, unless HALT_ON_ERROR is set, in which case it will fail.
Reset the members list by calling the reset_members method of ColorCorrection or reset all class member list and dictionaries with cdl_convert.reset_all.

Parsing a single correction CDL file

Instead of creating a blank CDL object, you can parse a cc file from disk, and it will return a single ColorCorrection matching the correction found in the file. Formats that contain multiple corrections will return a ColorCollection , which contains child ColorCorrection .

If you don't want to worry about matching the filetype to a parser, just use the generic parse_file function.

```
>>> cdl.parse_file('./myfirstcdl.cc')
<cdl_convert.correction.ColorCorrection object at 0x1004a5590>
>>> collection = cdl.parse_file('/myfirstedl.ccc')
<cdl_convert.collection.ColorCollection object at 0x100633b40>,
>>> collection.color_corrections
[
    <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
    <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
    <cdl_convert.correction.ColorCorrection object at 0x100633d90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b10>,
    <cdl_convert.correction.ColorCorrection object at 0x100633ad0>,
]
```

Once you have a `ColorCorrection` from a parser, you'll find that whatever values it found on the file now exist on the instance of `ColorCorrection`.

```
>>> cc = cdl.parse_cc('./xf/015.cc')
>>> cc.slope
(Decimal('1.02401'), Decimal('1.00804'), Decimal('0.89562'))
>>> cc.offset
(Decimal('-0.00864'), Decimal('-0.00261'), Decimal('0.03612'))
>>> cc.power
(Decimal('1.0'), Decimal('1.0'), Decimal('1.0'))
>>> cc.sat
Decimal('1.2')
>>> cc.id
'015_xf_seqGrade_v01'
>>> cc.file_in
'/Users/niven/cdls/xf/015.cc'
```

Note: When parsing, the `id` attribute is set in a variety of ways depending on how much information is available. Some formats, like `cc`, have an explicitly tagged `id` field that is always used. Other formats, like `flex`, have no such field and the parser tries to grab any scene/take metadata it can find to construct one. The last fallback is always the filename. For formats that can contain multiple `ColorCorrection` , the `id` has a created instance number after it.

4.2.2 Using `ColorCorrection`

Slope, Offset and Power

Setting the CDL slope, offset and power (SOP) values is as easy as passing them any list or tuple with three values. Integers, strings and floats will be automatically converted to Decimals, while slope and power will also truncate at zero.

```
>>> cc.slope = ('1.234', 5, 273891.37823)
>>> cc.slope
(Decimal('1.234'), Decimal('5.0'), Decimal('273891.37823'))
>>> cc.offset = (-0.0013, 0.097, 0.001)
>>> cc.offset
(Decimal('-0.0013'), Decimal('0.097'), Decimal('0.001'))
>>> cc.power = (-0.01, 1.0, 1.0)
>>> cc.power
(Decimal('0.0'), Decimal('1.0'), Decimal('1.0'))
```

```
>>> cc.power = (1.01, 1.007)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "cdl_convert/correction.py", line 306, in power
    self.sop_node.power = power_rgb
  File "cdl_convert/correction.py", line 668, in power
    value = self._check_setter_value(value, 'power')
  File "cdl_convert/correction.py", line 767, in _check_setter_value
    value = self._check_rgb_values(value, name, negative_allow)
  File "cdl_convert/correction.py", line 709, in _check_rgb_values
    values=values
ValueError: Error setting power with value: "(1.01, 1.007)". Power values given as a list or tuple mu
```

It's also possible to set the SOP values with a single value, and have it copy itself across all three colors. Setting SOP values this way mimics how color corrections typically start out.

```
>>> cc.slope = 1.2
>>> cc.slope
(Decimal('1.2'), Decimal('1.2'), Decimal('1.2'))
```

Saturation

Saturation is a positive float values, and the same checks and conversions that we do on SOP values happen for saturation as well.

```
>>> cc.sat = 1.1
>>> cc.sat
Decimal('1.1')
>>> cc.sat = '1.2'
>>> cc.sat
Decimal('1.2')
>>> cc.sat = 1
>>> cc.sat
Decimal('1.0')
>>> cc.sat = -0.1
>>> cc.sat
Decimal('0.0')
```

Warning: If it's desired to have negative values raise an exception instead of truncating to zero, set the global config module variable `HALT_ON_ERROR` to be `True`.

```
>>> cdl.config.HALT_ON_ERROR = True
>>> cc.power = (-0.01, 1.0, 1.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "cdl_convert/correction.py", line 306, in power
    self.sop_node.power = power_rgb
  File "cdl_convert/correction.py", line 668, in power
    value = self._check_setter_value(value, 'power')
  File "cdl_convert/correction.py", line 767, in _check_setter_value
    value = self._check_rgb_values(value, name, negative_allow)
  File "cdl_convert/correction.py", line 720, in _check_rgb_values
    negative_allow
  File "cdl_convert/base.py", line 419, in _check_single_value
    value=value
ValueError: Error setting power with value: "-0.01". Values must not be negative
```

Description

Certain formats of the cdl will contain multiple description entries. Each description entry is added to the `desc` attribute, which returns a list of the entries.

```
>>> cc.desc
['John enters the room', '5.6 ISO 800', 'bad take']
```

You can append to list by setting the description field like normal.

```
>>> cc.desc = 'final cc'
>>> cc.desc
['John enters the room', '5.6 ISO 800', 'bad take', 'final cc]
```

Setting the value to a new list or tuple will replace the list.

```
>>> cc.desc
['John enters the room', '5.6 ISO 800', 'bad take', 'final cc]
>>> cc.desc = ['first comment', 'second comment']
>>> cc.desc
['first comment', 'second comment']
```

Id and Files

When creating a `ColorCorrection`, the `id` field is checked against a global list of `ColorCorrection` ids, and creation fails if the `id` is not unique.

You can change the id after creation, but it will perform the same check.

```
>>> cc = cdl.ColorCorrection(id='cc1', input_file='./myfirstcdl.cc')
>>> cc2 = cdl.ColorCorrection(id='cc2', input_file='./mysecondcdl.cc')
>>> cc.id
'cc1'
>>> cc2.id
'cc2'
>>> cc2.id = 'cc1'
Traceback (most recent call last):
  File "<ipython-input-8-b2b5487dbc63>", line 1, in <module>
    cc2.id = 'cc1'
  File "cdl_convert/correction.py", line 362, in id
    self._set_id(value)
  File "cdl_convert/correction.py", line 430, in _set_id
    cc_id=cc_id
ValueError: Error setting the id to "cc1". This id is already a registered id.
```

A ValueError is only raised if `HALT_ON_ERROR` is set. If `HALT_ON_ERROR` is not set (default), a number will be appended to the non-duplicate ID.

So if you already have a ColorCorrection with the id of 'sh100cc', the second ColorCorrection you set to have that id will actually set to 'sh100cc001'.

At the current time, `file_out` cannot be set directly. `file_out` is determined by using the class method `determine_dest`, which takes a provided directory, the `id` and figures out the output path.

```
>>> cc.file_in
'/Users/sean/cdls/xf/015.cc'
>>> cc.file_out
>>> cc.determine_dest('cdl', '/Users/potter/cdls/converted/')
>>> cc.id
```

```
'015_xf_seqGrade_v01'
>>> cc.file_out
'/Users/potter/cdls/converted/015_xf_seqGrade_v01.cdl'
```

4.2.3 Writing `ColorCorrection`

When you're done tinkering with the `ColorCorrection` instance, you might want to write it out to a file. We need to give `ColorCorrection` the file extension we plan to write to, then call a `write` function with our `ColorCorrection` instance, which will actually convert the values on the `ColorCorrection` into the format desired, then write that format to disk.

```
>>> cc.determine_dest('cdl', '/Users/potter/cdls/converted/')
>>> cc.file_out
'/Users/potter/cdls/converted/015_xf_seqGrade_v01.cdl'
>>> cdl.write_cdl(cc)
```

Warning: It is highly likely that in the future, these will be methods on the `ColorCorrection` class itself, and that instead of writing the file directly, they will instead return a string formatted for writing.

4.3 Color Collections

Once installed with pip, importing `cdl_convert` works like importing any other python module.

```
>>> import cdl_convert as cdl
```

4.3.1 Creating `ColorCollection`

The `ColorCollection` class represents both the `ColorCorrectionCollection` and `ColorDecisionList` containers of the ASC CDL spec.

The distinctions between the two are fairly trivial:

`ColorCorrectionCollection` contain one or more `ColorCorrections` (which directly correspond to `ColorCorrection`), as well as the normal `Description`, `InputDescription` and `ViewingDescription` fields.

`ColorDecisionList` contain `ColorDecision` (directly corresponding to `ColorDecision`) instead of `ColorCorrection` . Those `ColorDecision` in turn contain the same `ColorCorrection` elements that `ColorCorrectionCollection` directly contains. Alongside the `ColorCorrection` are optional `MediaRef` elements (again directly corresponding to `MediaRef`), which simply contain a path to reference media for the `ColorCorrection` alongside.

Note: One final difference is that instead of a `ColorCorrection` element, a `ColorDecision` could instead contain a `ColorCorrectionRef`, which is simply an `id` reference to another "ColorCorrection.

`ColorCollection` has a `type` attribute that determines what the `ColorCollection` currently describes when you call its XML attributes. Setting this to `'ccc'` will cause a `ColorCorrectionCorrection` to be returned when the `xml` attribute is retrieved. Setting it to `'cdl'` causes a `ColorDecisionList` to appear instead.

Note: No matter what combination of `ColorDecision` or `ColorCorrection` a single `ColorCollection` has, any members of the 'opposite' class will be displayed correctly when you switch the `type`.

If you have 3 `ColorDecision` (each with their own `ColorCorrection`) under the `color_decisions` attribute, and 4 `ColorCorrection` under the `color_corrections` attribute, the XML will export 7 `ColorCorrection` elements when `type` is set to `'ccc'`, and 7 `ColorDecision` elements when `type` is set to `'cdl'`.

The converted elements are created 'on the fly' and are not saved, simply exported that way.

Unlike a `ColorCorrection` , `ColorCollection` does not have any default values. The description attributes it inherits from `AscColorSpaceBase` and `AscDescBase` default to none.

Those inherited attributes are `input_desc` (to describe the colorspace entering the correction, `viewing_desc` (to describe the colorspace conversions that must occur for viewing, and what type of monitor was used), and `desc` (which can be an infinitely long list of shot descriptions)

Note: When a child `ColorCorrection` **does not** have an `input_desc` or a `viewing_desc` of it's own and that child is exported alone to a `.cc` file, the descriptions from it's parent are used.

When a child `ColorCorrection` **has** an `input_desc` or a `viewing_desc`, that attribute is considered to have overruled the parent attribute.

In both cases, `desc''s from the parent are prepended to the child node's ''desc`.

When elements (such as `desc`) are placed into the child `ColorCorrection`, their text data is prepended with `From Parent Collection:` to easily distinguish between inherited fields and native.

Warning: The above note describes behavior not yet implemeneted and should be ignored. The author of the above note has been sacked.

Direct Creation

Creating a new `ColorCollection` is easy, and requires no arguments.

```
>>> ccc = cdl.ColorCollection()
```

Alternatively, you can pass in an `input_file`:

```
>>> ccc = cdl.ColorCollection(input_file='CoolMovieSequence.ccc')
>>> ccc.file_in
'/proj/UltimateMovie/share/color/CoolMovieSequence.ccc'
```

Parsing a CDL Collection file

When the collection you want to manipulate already exists, you'll want to parse the file on disk. EDL files, `.ccc` and `.cdl` files all return a single `ColorCollection` object, which contains all the child color corrections.

```
>>> collection = cdl.parse_ccc('/myfirstedl.ccc')
<cdl_convert.ColorCollection object at 0x100633b40>,
>>> collection.color_corrections
[
    <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
```

```
    <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
    <cdl_convert.correction.ColorCorrection object at 0x100633d90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b10>,
    <cdl_convert.correction.ColorCorrection object at 0x100633ad0>,
]
```

When parsing to a `ColorCollection` from disk, the type of file you parse determines what `type` is set to. Parsing an EDL or a `.cdl` file creates a `ColorCollection` with a type of `'cdl'` (since EDLs contain many media references and may even include `ColorCorrectionRef` elements), while parsing a `.ccc` file or multiple `.cc` files will create an instance with a type of `'ccc'`.

Note: At the current time, parsing EDLs results on a `ccc` collection, not a `cdl` as stated above.

4.3.2 Using `ColorCollection`

Adding children to `ColorCollection`

Already created `ColorCorrection` or `ColorDecision` can be added to the correct child list by calling the `append_child` method.

```
>>> ccc.color_corrections
[]
>>> ccc.append_child(cc)
>>> ccc.color_corrections
[
    <cdl_convert.correction.ColorCorrection object at 0x1004a5590>
]
>>> ccc.append_child(cd)
>>> ccc.color_decisions
[
    <cdl_convert.decision.ColorDecision object at 0x1004a5510>
]
```

`append_child` automatically detects which type of child you are attempting to append, and places it in the correct list. You can use `append_children` to append a list of children at once- the list can even contain mixed classes.

```
    >>> list_of_colors
    [
        <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
        <cdl_convert.decision.ColorDecision object at 0x100633b10>,
        <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
        <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
        <cdl_convert.decision.ColorDecision object at 0x100633d90>,
        <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
        <cdl_convert.decision.ColorDecision object at 0x100633ad0>,
    ]
    >>> ccc.append_children(list_of_colors)
    >>> ccc.color_corrections
    [
        <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
        <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
        <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
        <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
    ]
```

```
>>> ccc.color_decisions
[
    <cdl_convert.decision.ColorDecision object at 0x100633d90>,
    <cdl_convert.decision.ColorDecision object at 0x100633b10>,
    <cdl_convert.decision.ColorDecision object at 0x100633ad0>,
]
```

`append_child` and `append_children` will fail if you attempt to append a child which has a matching `id` to an already present child. The only exception is a `ColorCorrectionRef` , which of course should have the same `id` as a full `ColorCorrection` .

Warning: Both `appand_child` and `append_children` will change the `parent` attribute of `ColorCorrection` and `ColorDecision` to point to the `ColorCollection` they are appending to. Since we don't enforce a 1 parent to each child relationship, it's very easy to accidentally lose track of original parentage. While the child's `parent` attribute might point to another `ColorCollection`, the children of a collection will never be removed from the `color_corrections`, `color_decisions` and `all_children` lists.
You can immediately reset the `parent` attribute to point to a specific instance of `ColorCollection` by calling the `set_parentage` method.

Merging multiple `ColorCollection`

If you have multiple `ColorCollection` and wish to end up with a single collection, you'll need to merge them together. Assuming you have two `ColorCollection` with the names `ccc` and `dl` with the following information:

```
>>> ccc.input_desc
'LogC to sRGB'
>>> ccc.viewing_desc
'DaVinci Resolve on Eizo'
>>> ccc.desc
[
    'When Babies Attack Test DI',
    'Do not use for final',
    'Color by Zap Brannigan',
]
>>> ccc.type
'ccc'
>>> ccc.all_children
[
    <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
    <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
]
>>> dl.input_desc
'Cineon Log'
>>> dl.viewing_desc
'Panasonic Plasma rec709'
>>> dl.desc
[
    'Animals shot with a fisheye lens',
    'cute fluffy animals',
    'watch for blown out highlights',
    'Color by Zap Brannigan',
]
>>> dl.type
'cdl'
```

```
>>> dl.all_children
[
    <cdl_convert.decision.ColorDecision object at 0x100633d90>,
    <cdl_convert.decision.ColorDecision object at 0x100633b10>,
    <cdl_convert.decision.ColorDecision object at 0x100633ad0>,
]
```

You merge by choosing a 'parent' collection, and calling the `merge_collections` method on it.

```
>>> merged = ccc.merge_collections([dl])
>>> merged.all_children
[
    <cdl_convert.correction.ColorCorrection object at 0x100633b90>,
    <cdl_convert.correction.ColorCorrection object at 0x100633c50>,
    <cdl_convert.correction.ColorCorrection object at 0x100633cd0>,
    <cdl_convert.correction.ColorCorrection object at 0x100633b50>,
    <cdl_convert.decision.ColorDecision object at 0x100633d90>,
    <cdl_convert.decision.ColorDecision object at 0x100633b10>,
    <cdl_convert.decision.ColorDecision object at 0x100633ad0>,
]
```

Note: When merging multiple `ColorCollection` , any duplicate children objects (if you had the same `ColorCorrection` object assigned as a child to multiple `ColorCollection`) are removed, so the list only contains unique members.

The parent determines which Input and Viewing Description overrides all of the other merged collections. `type` is also set to match the `type` of the parent. Since `ccc` was our parent:

```
>>> merged.input_desc
'LogC to sRGB'
>>> merged.viewing_desc
'DaVinci Resolve on Eizo'
>>> merged.type
'ccc'
```

If we had used `dl` as the merged parent:

```
>>> merged = dl.merge_collections([ccc])
>>> merged.input_desc
'Cineon Log'
>>> merged.viewing_desc
'Panasonic Plasma rec709'
>>> merged.type
'cdl'
```

Unlike the Input and Viewing Descriptions, the normal Description attributes are all merged together.

```
>>> merged.desc
[
    'When Babies Attack Test DI',
    'Do not use for final',
    'Color by Zap Brannigan',
    'Animals shot with a fisheye lens',
    'cute fluffy animals',
    'watch for blown out highlights',
    'Color by Zap Brannigan',
]
```

Note: Unlike the lists of children, duplicates are not removed from the list of descriptions.

4.4 Installation

4.4.1 Pip Install

Using pip is the preferred method of installation, as it installs cdl_convert both as a python module and a script.

```
$ pip install cdl_convert
```

4.4.2 Script Only Installation

If you don't want to bother with a pip style install, you can alternatively grab the entire cdl_convert directory, then set up a shortcut to call `cdl_convert/cdl_convert.py`

Creating aliases, etc are beyond the scope of this documentation.

4.5 Changelog

4.5.1 Version 0.9.2

- Fixed a bug where ALE's with blank lines would not convert correctly.
- Fixed a bug that was preventing `cdl_convert` from being correctly installed in Python 2.6
- Fixed continuous integration testing.
- No longer officially supporting Python 3.2, as I've had to remove it from our CI builds. It should still work just fine though, but we won't be running CI against it.

4.5.2 Version 0.9

- Added ability to parse CMX EDLs
- Fixed a script bug where a collection format containing color decisions will not have those color decisions exported as individual color corrections.
- Fixed a bug where we weren't reading line endings correctly in certain situations.
- Added a cdl_convert.py stub file to the package root level, which will allow running of the cdl_convert script without installation. Due to relative imports in the python code, it was no longer possible to call cdl_convert/cdl_convert.py directly.
- The script, when run directly from cdl_convert.py, will now write errors to stderror correctly, and exit with a status of 1.

4.5.3 Version 0.8

- Added *–single* flag. When provided with an output collection format, each color correction in the input will be exported to it's own collection.
- Giving a ColorCorrection a non-duplicate ID now works unless the `--halt` flag is given. This means that incoming collections that contain duplicate IDs will not fail out.

4.5.4 Version 0.7.1

- Fixed bug where ALE's without 'Scan Filename' fields could not parse correctly.

4.5.5 Version 0.7

The biggest change in 0.7 is the addition of collection format support. `.ccc`, Color Correction Collections, can now be parsed and written. `.cdl`, Color Decision Lists, can now be parsed and written. `.ale` and `.flex` files now return a collection.

- **New script flags:**
  - Adds `--check` flag to script, which checks all parsed `ColorCorrects` for sane values, and prints warnings to shell
  - Adds `-d`, `--destination` flag to the script, which allows user to specify the output directory converted files will be written to.
  - Adds `--no-ouput` flag to the script, which goes through the entire conversion process but doesn't actually write anything to disk. Useful for troubleshooting, especially when combined with `--check`
  - Adds `--halt` flag to the script, which halts on errors that can be resolved safely (such as negative slope or power values)
- Renames `ColorCollectionBase` to `ColorCollection` , since it will be used directly by both `ccc` and `cdl`.
- Adds `parse_ccc` which returns a `ColorCollection` .
- Adds `write_ccc` which writes a `ColorCollection` as a `ccc` file.
- Adds `parse_cdl` which returns a `ColorCollection` .
- Adds `write_cdl` which returns a `ColorCollection` as a `cdl` file.
- `ColorCollection` is now a fully functional container class, with many attributes and methods.
- Added `ColorDecision` , which stores either a `ColorCorrection` or `ColorCorrectionRef` , and an optional `MediaRef`
- Added `ColorCorrectionRef` , which stores a reference to a `ColorCorrection`
- Added `parent` attribute to `ColorCorrection` .
- Calling `sop_node` or `sat_node` on a `ColorCorrection` before attempting to set a SOP or Sat power now works.
- `ColorCorrection` `cdl_file` init argument renamed to `input_file`, which is now optional and able to be set after init.
- `parse_cc` and `parse_rnh_cdl` now only yield a single `ColorCorrection` , not a single member list.
- Added dev-requirements.txt (contains `mock`)

- All `determine_dest` methods now take a second `directory` argument, which determines the output directory.
- Adds `sanity_check` function which prints values which might be unusual to stdout.
- `parse_cdl` and `write_cdl` renamed to `parse_rnh_cdl` and `write_rnh_cdl` respectively.
- **`member_reset` methods:**
  - `ColorCorrection` now has a `reset_members` method, which resets the class level member's dictionary.
  - `MediaRef` also has a `reset_members` method, as does `ColorCollection`
  - `reset_all` function calls all of the above `reset_members` methods at once.
- **Renamed `cdl_file` argument:**
  - `parse_cc` `cdl_file` arg renamed to `input_file` and now accepts a either a raw string or an `ElementTree` `Element` as `input_file`.
  - `parse_rnh_cdl` `cdl_file` arg renamed to `input_file`.
  - `parse_ale` `edl_file` arg renamed to `input_file`.
  - `parse_flex` `edl_file` arg renamed to `input_file`.
- **Python Structure Refactoring**
  - Moved `HALT_ON_ERROR` into the `config` module, which should now be referenced and set by importing the entire `config` module, and referencing or setting `config.HALT_ON_ERROR`
  - Script functionality remains in `cdl_convert.cdl_convert`, but everything else has been moved out.
  - `AscColorSpaceBase` , `AscDescBase` , `AscXMLBase` and `ColorNodeBase` now live under `cdl_convert.base`
  - `ColorCollection` now lives in `cdl_convert.collection`
  - `ColorCorrection` , `SatNode` and `SopNode` now live under `cdl_convert.correction`
  - `ColorDecision` , `ColorCorrectionRef` and `MediaRef` now live under `cdl_convert.decision`
  - All parse functions now live under `cdl_convert.parse`
  - All write functions now live under `cdl_convert.write`
  - `sanity_check` now live under `cdl_convert.utils`
  - `reset_all` now lives under the main module

4.5.6 Version 0.6.1

- Added `AscXMLBase` class for nodes that can be represented by XML to inherit.
- Suppressed scientific notation from being written out when writing files. Should now write out as close as Python accuracy allows, and the same number of digits.
- `write_cc` now writes out 100% correct XML using ElementTree.
- Added tests for `write_cc`, which **brings our coverage to 100%**

4.5.7 Version 0.6

- Adds much greater ASC CDL XML compliance with the addition of many classes that represent node concepts in the CDL XML schema.
- Moves `viewing_desc` and `input_desc` attributes and methods into the base class `AscColorSpaceBase` .
- Moved `desc` attribute and methods into the base class `AscDescBase` .
- Adds `ColorCollectionBase` class for a basis of all collection type nodes (`ColorCorrectionCollection` , `ColorDecisionList` , etc).
- Adds `MediaRef` class which represents the MediaRef node of a ColorDecision. This class allows convenient handling of files given as media reference.
- Adds `HALT_ON_ERROR` module variable which determines certain exception handling behavior. Exceptions that can normally be handled with default behavior (such as negative Slope or Power values) will be dealt with silently instead of stopping the program. Negative Slope and Power values, for example, will clip to 0.0.
- **`ColorCorrection` (formally `AscCdl`) class changes:**
  - Renames `AscCdl` to `ColorCorrection` .
  - Adds class level member dictionary, which allows lookup of a `ColorCorrection` instance by the unique ID.
  - `ColorCorrection` objects now require a unique ID to be instantiated.
  - Removes `metadata` attribute of `ColorCorrection` .
  - Moves SOP and SAT operations out of `ColorCorrection` into their own classes, which are based on `ColorNodeBase` . The `SatNode` and `SopNode` classes are still meant to be children of `ColorCorrection`.
  - Added `sop_node` and `sat_node` attributes to access the child `SatNode` and `SopNode` .
  - Removed `metadata` attribute, splitting it into the inherited attributes of `input_desc`, `viewing_desc` and `desc`.
  - `desc` attribute is now fully fleshed out as a list of all encountered description fields.
  - Renamed `cc_id` field to `id`, shadowing the built in `id` within the class.
  - Slope, Offset and Power now return as a tuple instead of a list to prevent index assignment, appending and extending.
- **`parse_cc` should now parse a much greater variety of `.cc` files more accurately.**
  - Now supports infinite Description fields
  - Now supports Viewing and Input Description fields
  - Significantly simplifies the function.
- `parse_flex` has been significantly simplified.
- Test Suite broken up into sub-modules.
- Adds PyPy support.
- Adds ReadTheDocs
- Adds docs to build

4.5.8 Version 0.5

- Project is now structured according to Python packaging guidelines with `setup.py` etc.
- Some `AscCdl` attributes have been moved into dictionaries (Note that this was later reversed in release 0.6)
- Refactors some parse functions to be less complex
- Makes `write_cdl` much simpler and more pythonic.

4.5.9 Version 0.4.2

- Hotfix to fix `from __future__` imports

4.5.10 Version 0.4.1

- PEP 8 conversion
- landscape.io support
- Uses `from __future__` for print

4.5.11 Version 0.4

- Python 3 compatible
- More unit testing bug fixes and enhancements.
- Adds better type and exception handling for `AscCdl` setters.
- Now sanitizes id fields of any characters they shouldn't contain.
- Test suite runs on windows now
- Adds Travis-ci for continuous integration testing
- `parse_cc` now uses `ElementTree` for XML parsing

4.6 Support

4.6.1 GitHub

At `cdl_convert`'s GitHub page you can browse the code and the history of the project.

Builds can be downloaded from the GitHub page or the PyPI repository entry.

4.6.2 Bug Reporting

The issues page on GitHub is the best place to report bugs or request support, and while `cdl_convert` is distributed with no warranty of any kind, issues will be read and helped if able.

Please fill out an issue describing the problem you are having. Attaching sample files to show what's not working, and the full printout from your shell.

4.6.3 Contact

Support will not be given over email, twitter, etc. If you need support, use the issues page at GitHub. That said, if you want to say 'hi' or see what I'm currently working on, you can try me at one of the following:

- twitter: @shidarin
- email: shidarin@alphamatte.com
- github blog: http://shidarin.github.io/

4.7 Frequently Asked Questions

4.7.1 Python 2 & 3 Support

- **What versions of Python does `cdl_convert` support?** `cdl_convert` works in Python 2.6 through 3.4 and PyPy. A full test suite runs continuous integration through Travis-ci.org, coverage through coveralls.io, and code quality checked with landscape.io. **Code is PEP 8 compliant**, with docstrings following google code docstring standards.
- **Really? It works on both Python 2 and 3? And PyPy?** Yes. No conversion or modification needed.

4.7.2 CDL Format Support

- **Why don't you support format *X*?** I either haven't had time to build a parser for the format yet, or I might even be unaware it exists. Perhaps you should drop by the issues page and create a request for the format? If creating a request for a format it helps immensely to have a sample of that format.

4.7.3 Project Structure

- **Why the underscore?** `cdl_convert` started as a simple script to convert from one format to another. As such, it wasn't named with the standards that one would usually use for a python module. By the time the project became big enough, was on PyPI, etc, it was too spread out on the web, in too many places to make changing easy. In the end, I opted to keep it. At some point, `cdl_convert` might migrate into a larger, more generic film & tv python module, which will be named properly.

4.8 Contributing

Note: Portions of this page have been modified from the excellent OpenComparison project docs.

4.8.1 Contributing CDL and EDL Samples

Please, *please*, **please** submit samples of the following formats:

- FLEx
- ALE
- CMX

- CCC
- CDL

These are complex formats, and seeing real world samples helps write tests that ensure correct parsing of real world EDLs and CDLs. If you don't even see a format of CDL listed that you know exists, open an issue at the github issues page asking for parse/write support for the format, and include a sample.

4.8.2 Issues & Bugs

Take a look at the issues page and if you see something that you think you can bang out, leave a comment saying you're going to take it on. While many issues are already assigned to the principal authors, just because it's assigned doesn't mean any work has begun.

Feel welcome to post issues, feature requests and bugs that aren't present.

4.8.3 Workflow

`cdl_convert` is a GitFlow workflow project. If you're not familiar with GitFlow, please take a moment to read the workflow documentation. Essentially it means that all work beyond tiny bug fixes needs to be done on it's own feature branch, called something like `feature/thing_I_am_fixing`.

After you fork the repository, take a second to create a new feature branch from the `develop` branch and checkout that newly created branch.

Submitting Your Fix

Once you've pushed your feature branch to GitHub, it's time to generate a pull request back to the central `cdl_convert` repository.

The pull request let's you attach a comment, and when you're addressing an issue it's imperative to link to that issue in the initial pull request comment. We'll shortly be notified of your request and it will be reviewed as soon as possible.

> **Warning:** If you continue to add commits to the feature branch you submitted as a pull request, the pull request will be updated with those changes (as long as you push those changes to GitHub). This is why you should not submit a pull request of the `develop` branch.

4.8.4 Pull Request Tips

`cdl_convert` really needs your collaboration, but we only have so much time to work on the project and merge your fixes and features in. There's some easy to follow guidelines that will ensure your pull request is accepted and integrated quickly.

Run the tests!

Before you submit a pull request, please run the entire test suite via

```
$ python setup.py test
```

If the tests are failing, it's likely that you accidentally broke something. Note which tests are failing, and how your code might have affected them. If your change is intentional- for example you made it so urls all read `https://` instead of `http://`, adjust the test suite, get it back into a passing state, and then submit it.

If your code fails the tests (Travis-ci.org checks all pull requests when you create them) it will be **rejected**.

Add tests for your new code

If your pull request adds a feature but lacks tests then it will be **rejected**.

Tests are written using the standard unittest framework. Please keep test cases as simple as possible while maintaining a good coverage of the code you added.

> **Warning:** Tests are currently written in the style of unittest with camelCased method & variable names. Please follow **PEP 8** otherwise.

Don't mix code changes with whitespace cleanup

If you change two lines of code and correct 200 lines of whitespace issues in a file the diff on that pull request is functionally unreadable and will be **rejected**. Whitespace cleanups need to be in their own pull request.

Keep your pull requests limited to a single issue

Pull requests should be as small/atomic as possible. Large, wide-sweeping changes in a pull request will be **rejected**, with comments to isolate the specific code in your pull request.

Follow PEP-8 and keep your code simple!

Memorize the Zen of Python

```
>>> python -c 'import this'
```

Please keep your code as clean and straightforward as possible. When we see more than one or two functions/methods starting with *_my_special_function* or things like *__builtins__.object = str* we start to get worried. Rather than try and figure out your brilliant work we'll just **reject** it and send along a request for simplification.

Furthermore, the pixel shortage is over. We want to see:

- *package* instead of *pkg*
- *grid* instead of *g*
- *my_function_that_does_things* instead of *mftdt*

If the code style doesn't follow **PEP 8** , it's going to be **rejected**.

4.8.5 Copyright of Submitted Contributions

When submitting, you'll be asked to waive copyright to your submitted code to the listed authors. This is so we can keep a tight handle on the code and change the license for future releases if needed.

4.9 Licenses

4.9.1 Software

The MIT License (MIT)

4.9.2 Documentation

cdl_convert

OpenComparison

4.10 API Reference

4.10.1 Classes

The class structure of `cdl_convert` mirrors the element structure of the XML schema for `ccc`, `cdl` and `cc` files as defined by the

ASC. The XML schema's represent the most complicated and structured variant of the format, and therefore it behooves the project to mimic their structure.

However, where similar elements exist as entirely separate entities in the XML schema, they might have been combined here.

AscColorSpaceBase

Classes that deal with input and viewer colorspace can subclass from this class to get the `input_desc` and `viewing_desc` attributes.

class `cdl_convert.base.`**`AscColorSpaceBase`**

Base class for Asc XML type nodes that deal with colorspace

This class is meant to be inherited by any node type that used viewing and input colorspace descriptions.

This class doesn't do a lot right now, as we don't have any specific controls on how to set or retrieve these fields. In the future however, we'll parse incoming descriptions to try and resolve input colorspace and viewing colorspace.

Attributes:

input_desc [(str)] Description of the color space, format and properties of the input images. Individual `ColorCorrections` can override this.

viewing_desc [(str)] Viewing device, settings and environment. Individual `ColorCorrections` can override this.

Public Methods:

parse_xml_input_desc() Parses an ElementTree Element to find & add an InputDescription. If none is found, `input_desc` will remain set to `None`.

parse_xml_viewing_desc() Parses an ElementTree Element to find & add a ViewingDescription. If none is found, `viewing_desc` will remain set to `None`.

AscDescBase

Classes that are allowed to have a description field subclass from this from this class to get the `desc` attribute. The `desc` attribute can be set with a single string, which will append to the list of strings already present in `desc`. If set to a list or tuple, `desc` will become a list of those values. If set to `None`, `desc` will become an empty list.

class `cdl_convert.base.`**`AscDescBase`**

Base class for most Asc XML type nodes, allows for infinite desc

This class is meant to be inherited by any node type that uses description fields.

Attributes:

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Public Methods:

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`.

AscXMLBase

class `cdl_convert.base.`**`AscXMLBase`**

Base class for nodes which can be converted to XML Elements

This class contains several convenience attributes which can be used to retrieve ElementTree Elements, or nicely formatted strings.

Attributes:

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node.

xml [(str)] A nicely formatted XML string representing the node.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line.

Public Methods:

build_element() A placeholder method to be overridden by inheriting classes, calling it will always return None.

ColorCollection

This class functions as both a ColorDecisionList and a ColorCorrectionCollection. It's children can be either ColorDecisions, ColorCorrections, or a combination of the two. Despite being able to have either type of child, the `ColorCollection` still needs to know which type of collection you want it to represent.

Setting the `type` of the `ColorCollection` to either `ccc` or `cdl` causes children of the opposite type to be converted into the appropriate type when exporting the class.

Note: `parse_ale` and `parse_flex` both return as a `ccc` at this time, contrary to the documentation below.

In addition, the inclusion of parent metadata into orphaned children is also a work in progress.

If parse_ale is used to parse an ale edl file, the ale will be read into a ColorCollection set to cdl and the children the ale creates will actually be ColorDecision , as that allows for the easy inclusion of MediaRef objects. If you then use write_ccc to write a ccc file, all the children ColorDecision will create XML elements for their ColorCorrection children, adding in any MediaRef that were found as Description elements. Finally the ColorCollection type is set to ccc and the xml_root field is called, which knows to return a ccc style XML element to write_ccc.

class cdl_convert.collection.**ColorCollection**(*input_file=None*)

Container class for ColorDecisionLists and ColorCorrectionCollections.

Collections need to store children and have access to descriptions, input descriptions, and viewing descriptions.

Class Attributes:

members [[:class'ColorCollection']] All instanced ColorCollection are added to this member list. Unlike the ColorCorrection member's dictionary, ColorCollection do not need any unique values to exist.

This is currently only used for determining an id value when exporting and no file_in attribute is set.

Attributes:

all_children [(ColorCorrection, ColorDecision)] A tuple of all the children of this collection, both Corrections and Decisions.

color_corrections [(ColorCorrection)] All the ColorCorrection children are listed here.

color_decisions [(ColorDecision)] All the ColorDecision children are listed here.

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from AscDescBase .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from AscXMLBase .

file_in [(str)] Filepath used to create this ColorCollection .

file_out [(str)] Filepath this ColorCollection will be written to.

input_desc [(str)] Description of the color space, format and properties of the input images. Inherited from AscColorSpaceBase .

is_ccc [(bool)] True if this collection currently represents .ccc.

is_cdl [(bool)] True if this collection currently represents .cdl.

type [(str)] Either ccc or cdl, represents the type of collection this class currently will export by default.

viewing_desc [(str)] Viewing device, settings and environment. Inherited from AscColorSpaceBase .

xml [(str)] A nicely formatted XML string representing the node. Inherited from AscXMLBase.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from AscXMLBase.

xmlns [(str)] Describes the version of the ASC XML Schema that cdl_convert writes out to files following the full schema (`.ccc` and `.cdl`)

Public Methods:

append_child() Appends the given object, either a `ColorCorrection` or a `ColorDecision`, to the respective attribute list, either `color_corrections` or `color_decision` depending on the class of the object passed in.

append_children() Given a list, will iterate through and append each element of that list to the correct child list, using the `append_child()` method.

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase`.

Here on `ColorCollection`, this is a pointer to `build_element_ccc()` or `build_element_cdl()` depending on which type the `ColorCollection` is currently set to.

build_element_ccc() Builds a CCC style XML tree representing this `ColorCollection` instance.

build_element_cdl() Builds a CDL style XML tree representing this `ColorCollection` instance.

copy_collection() Creates and returns an exact new instance that's an exact copy of the current instance. Note that references to the child instances will be copied, but that the child instances themselves will not be.

merge_collections() Merges all members of a list containing `ColorCollection` and the instance this is called on to return a new `ColorCollection` that is primarily a copy of this instance, but contains all children and description elements from the given collections. *input_desc*, *viewing_desc*, *file_in*, and *type* will be set to the values of the parent instance.

parse_xml_color_corrections() Parses an ElementTree element to find & add all ColorCorrection.

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

parse_xml_input_desc() Parses an ElementTree Element to find & add an InputDescription. If none is found, `input_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

parse_xml_viewing_desc() Parses an ElementTree Element to find & add a ViewingDescription. If none is found, `viewing_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

reset_members() Resets the class level members list.

set_parentage() Sets all child `ColorCorrection` and `ColorDecision` `parent` attribute to point to this instance.

set_to_ccc() Switches the `type` of this collection to export a `ccc` style xml collection by default.

set_to_cdl() Switches the `type` of this collection to export a `cdl` style xml collection by default.

ColorCorrection

The `ColorCorrection` class is the backbone of cdl_convert, as it represents the basic level of the color decision list concept- the color decision list itself. All the parse functions exist to extract the CDL metadata and populate this class, and all the write functions exist to write files out from this class.

Parser –> `ColorCorrection` –> Writer

`ColorCorrection` can of course be instantiated and used without a parse function, see Script Usage for a walk-through.

Warning: When an instance of `ColorCorrection` is first created, the `id` provided is checked against a class level dictionary variable named `members` to ensure that no two `ColorCorrection` share the same `id` , as this is required by the specification.
If the `id` does match an already created `id` and `HALT_ON_ERROR` is not set, the `id` assignment will go forward, appending the duplicate number to the back of the `id`. So the 2nd instance of 'sh100cc' will become 'sh100cc001'.
Reset the members dictionary by either calling the `reset_members` method on `ColorCorrection` or by reseting all cdl_convert member lists and dictionaries with the `reset_all` function.
If the `id` given is a blank string and `HALT_ON_ERROR` is set to `False`, `id` will be set to the total number of `ColorCorrection` in the file, including the one currently being created. This behavior is not accepted when changing the `id` after creation.

Warning: `cdl_file` is likely to not be a required attribute in the future.

class `cdl_convert.correction.`**`ColorCorrection`**(*id*, *input_file=None*)

The basic class for the ASC CDL

This class contains attributes for all 10 color correction numbers needed for an ASC CDL, as well as other metadata like shot names that typically accompanies a CDL.

These names are standardized by the ASC and where possible the attribute names will follow the ASC schema. Descriptions for some of these attributes are paraphrasing the ASC CDL documentation. For more information on the ASC CDL standard and the operations described below, you can obtain the ASC CDL implementor-oriented documentation by sending an email to: asc-cdl at theasc dot com

Order of operations is Slope, Offset, Power, then Saturation.

Class Attributes:

members [{str: :class'ColorCorrection' }] All instanced `ColorCorrection` are added to this member dictionary, with their unique id being the key and the `ColorCorrection` being the value.

Attributes:

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from `AscDescBase` .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

file_in [(str)] Filepath used to create this `ColorCorrection` .

file_out [(str)] Filepath this `ColorCorrection` will be written to.

has_sat [(bool)] Returns True if SOP values are set

has_sop [(bool)] Returns True if SOP values are set

id [(str)] Unique XML URI to identify this CDL. Often a shot or sequence name.

Changing this value does a check against the cls.members dictionary to ensure the new id is open. If it is, the key is changed to the new id and the id is changed.

Note that this shadows the builtin id.

input_desc [(str)] Description of the color space, format and properties of the input images. Inherited from `AscColorSpaceBase` .

parent [(`ColorCollection`)] The parent node that contains this node.

sat_node [(`SatNode`)] Contains a reference to a single instance of `SatNode` , which contains the saturation value and descriptions.

sop_node [(`SopNode`)] Contains a reference to a single instance of `SopNode` , which contains the slope, offset, power values and descriptions.

viewing_desc [(str)] Viewing device, settings and environment. Inherited from `AscColorSpaceBase` .

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

determine_dest() When provided an output extension, determines the destination filename to be written to based on `file_in` & `id`.

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

parse_xml_input_desc() Parses an ElementTree Element to find & add an InputDescription. If none is found, `input_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

parse_xml_viewing_desc() Parses an ElementTree Element to find & add a ViewingDescription. If none is found, `viewing_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

reset_members() Resets the class level members list.

ColorCorrectionRef

class `cdl_convert.decision.`**`ColorCorrectionRef`**(*id*)

Reference marker to a full color correction

This is a fairly basic class that simply contains a reference to a full `ColorCorrection` . The `id` attribute must match the `id` attribute in order for this class to function fully.

When writing to a format that allows empty references (like `cdl`), the reference can write correctly without breaking. However, if writing to a format that does not support reference objects at all (like `ccc`), attempting to write an empty reference will result in a `ValueError` (if `HALT_ON_ERROR` is set to `True`, or simply skip past the reference entirely.

Class Attributes:

members [{str: [:class'ColorCorrectionRef']}] All instanced `ColorCorrectionRef` are added to this member dictionary. Multiple `ColorCorrectionRef` can share the same reference id, therefore for each reference id key, the members dictionary stores a list of `ColorCorrectionRef` instances that share that `id` value.

Attributes:

cc [(`ColorCorrection`)] If the stored reference resolves to an existing `ColorCorrection`, this attribute will return that node using the `resolve_reference` method. This attribute is the same as calling that method.

parent [(`ColorDecision`)] The parent `ColorDecision` that contains this node.

id [(str)] The `ColorCorrection` id that this reference refers to. If `HALT_ON_ERROR` is set to `True`, will raise a `ValueError` if set to a `ColorCorrection` `id` value that doesn't yet exist.

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

reset_members() Resets the class level members list.

resolve_reference() Attempts to return the `ColorCorrection` that this reference is supposed to refer to.

If `HALT_ON_ERROR` is set to `True`, resolving a bad reference will raise a `ValueError` exception. If not set, it will simply return None.

Otherwise (if the `id` attribute matches a known `ColorCorrection` `id`, the `ColorCorrection` will be returned.

ColorDecision

ColorDecision's are normally found only within `ColorCorrection` but this limitation of the ASC CDL schema is not enforced by cdl_convert.

class `cdl_convert.decision.`**`ColorDecision`**(*color_correct=None*, *media=None*)

Contains a media ref and a ColorCorrection or reference to CC.

This class is a simple container to link a `ColorCorrection` (or `ColorCorrectionRef`) with a `MediaRef` . The `MediaRef` is optional, the ColorCorrection is not. The ColorCorrection does not need to be provided at initialization time however, as `ColorDecision` provides an XML element parser for deriving one.

The primary purpose of a ColorDecision node is to associate a ColorCorrection node with one or more items of Media Reference.

Along with Media Reference, a ColorDecision can contain the normal type of input, viewer and description metadata.

Additional, it is the only node that can contain ColorCorrectionRef nodes, which link the same ColorCorrection to many different ColorDecisions (and thus, many different items of media reference)

An example containing a ColorCorrection node:

```
<ColorDecision>
    <MediaRef ref="http://www.theasc.com/foasc-logo2.png"/>
    <ColorCorrection id="ascpromo">
        <SOPNode>
            <Description>get me outta here</Description>
            <Slope>0.9 1.1 1.0</Slope>
            <Offset>0.1 -0.01 0.0</Offset>
            <Power>1.0 0.99 1.0</Power>
        </SOPNode>
    </ColorCorrection>
</ColorDecision>
```

But it can also contain just a reference:

```
<ColorDecision>
    <MediaRef ref="best/project/ever/jim.0100.dpx"/>
    <ColorCorrectionRef ref="xf45.x628"/>
</ColorDecision>
```

Class Attributes:

members [{str: [:class'ColorDecision']}] All instanced `ColorDecision` are added to this member dictionary. The key is the id or reference id of the contained `ColorCorrection` or `ColorCorrectionRef` Multiple `ColorDecision` can , therefore for each reference id key, the members dictionary stores a list of `ColorDecision` instances that share that `id` value.

Attributes:

cc [(`ColorCorrection` , `ColorCorrectionRef`)] Returns the contained ColorCorrection, even if it's a reference.

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from `AscDescBase` .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

input_desc [(str)] Description of the color space, format and properties of the input images. Inherited from `AscColorSpaceBase` .

is_ref [(bool)] True if contains a `ColorCorrectionRef` object instead of a `ColorCorrection`

media_ref [(`MediaRef`)] Returns the contained `MediaRef` or None.

parent [(`ColorDecisionList`)] The parent node that contains this node.

set_parentage() Sets child `ColorCorrection` (or `ColorCorrectionRef`) and `MediaRef` (if present) `parent` attribute to point to this instance.

viewing_desc [(str)] Viewing device, settings and environment. Inherited from `AscColorSpaceBase` .

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

parse_xml_color_correction() Parses a ColorDecision ElementTree Element for a ColorCorrection Element or a ColorCorrectionRef Element.

parse_xml_color_decision() Parses a ColorDecision ElementTree Element for metadata, then calls parsers for ColorCorrection and MediaRef.

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

parse_xml_input_desc() Parses an ElementTree Element to find & add an InputDescription. If none is found, `input_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

parse_xml_media_ref() Parses an ColorDecision Element for a MediaRef Element.

parse_xml_viewing_desc() Parses an ElementTree Element to find & add a ViewingDescription. If none is found, `viewing_desc` will remain set to `None`. Inherited from `AscColorSpaceBase`

reset_members() Resets the class level members list.

ColorNodeBase

This class only exists to be subclassed by `SatNode` and `SopNode` and should not be used directly.

class `cdl_convert.base.`**`ColorNodeBase`**

Base class for SOP and SAT nodes.

This class is meant only to be inherited by `SopNode` and `SatNode` and should not be used outside of those classes.

It inherits from both `AscDescBase` and `AscXMLBase` giving the child classes both `desc` and `xml` related functionality.

This class is also home to a private function which helps `SopNode` and `SatNode` perform type and value checks on incoming values.

Attributes:

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from `AscDescBase` .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

MediaRef

Media Ref's are normally found only inside of `ColorDecision`, which itself is found only inside of the `ColorDecisionList` collection. This isn't a restriction that `cdl_convert` explicitly enforces, but the parse and write functions will only be creating and writing found `MediaRef` objects following the rules.

Where possible when writing filetypes that don't support `MediaRef`, the information kept in `MediaRef` will be converted into description field metadata and preserved in that way.

Note: The above metadata preservation is not yet implemented.

`MediaRef` is meant to provide a convenient interface for managing and interpreting data stored in CDLs. You can change a broken absolute link directory to a relative link without touching the filename, or retrieve a full list of image sequences contained within a referenced directory.

class `cdl_convert.decision.`**`MediaRef`**(*ref_uri*, *parent=None*)

A directory of files or a single file used for grade reference

`MediaRef` is a container for an image path that should be referenced in regards to the color correction being performed. What that reference means must be further clarified, either through communication or Description fields.

Requires a `ref_uri` and an optional `parent` to instantiate.

An XML URI is usually a filepath to a directory or file, sometimes proceeded by a protocol (such as `http://`). **Note that many of the functions and methods described below do not function properly when given a URI with a protocol in front.**

The parent of a `MediaRef` should typically be a `ColorDecision` , and in fact the CDL specification states that no other container is allowed to contain a `MediaRef`. That restriction is not enforced in the python API.

Class Attributes:

members [{str: [`MediaRef`]}] All instances of `MediaRef` are added to this class level members dictionary, with the key being the full reference URI. Since it's possible that multiple `MediaRef` point to the same reference URI, the value returned is a list of `MediaRef` that all have a value of that same URI.

When you change a single `MediaRef` ref attribute, it removes itself from the old key's list, and adds itself to the new key's list. The old key is removed from the dictionary if this `MediaRef` was the last member.

Attributes:

directory [(str)] The directory portion of the URI, without the protocol or filename.

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

exists [(bool)] True if the path is present in the file system.

filename [(str)] The filename portion of the URI, without any protocol or directory.

is_abs [(bool)] True if `directory` is an absolute reference.

is_dir [(bool)] True if `path` points to a directory with no filename portion.

is_seq [(bool)] True if `path` points to an image sequence or a directory of image sequences. Image sequences are determined by files ending in a dot or underscore, followed by an integer, followed by the file extension. If the filename reference given already has pound padding or %d indication padding, this will also return true.

Valid image sequences:

- TCM100X_20140215.0001.exr
- Bobs Big_Score_2.jpg
- 2383-279873.67267_32t7634.63278623781638218763.exr
- 104fl.x034.######.dpx
- 104fl.x034_%06d.dpx

parent [(`ColorDecision`)] The parent that contains this `MediaRef` object. This should normally be a `ColorDecision` , but that is not enforced.

path [(str)] The directory joined with the filename via os.path.join(), if there is no filename, path is identical to `directory`. If there is no protocol, `path` is identicial to `ref`.

protocol [(str)] The URI protocol section of the URI, if any. This is the section that proceeds the '://' of any URI. If there is no '://' in the given URI, this is empty.

ref [(str)] The full URI reference which includes the protocol, directory and filename. If there is no protocol and no filename, `ref` is identical to `directory`.

seq [(str)] If `is_seq` finds that the filename or directory refers to one or more image sequences, `seq` will return the first found sequence in the form of filename.####.ext (or filename_####.ext if the sequence has an _ in front of the frame numbers). *Note that there may be more than one image sequence if* `ref` *points to a directory*. To get a list of all image sequences found, use `seqs`.

Only if a reference was given to us already in the form of `%d` padding will `seq` and `seqs` return a sequence filename with `%d` padding.

seqs [[str]] Returns all found sequences in a list. If `ref` points to a filename, this list will only contain one sequence. If `ref` points to a directory, all sequences found in that directory will be in this list.

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

reset_members() Resets the class level members list.

SatNode

Note: This class is meant only to be created by a `ColorCorrection` , and thus has the required arg of `parent` when instantiating it.

class `cdl_convert.correction.`**`SatNode`**(*parent*)

Color node that contains saturation data.

Class Attributes:

element_names [[str]] Contains a list of XML Elements that refer to this class for use in parsing XML files.

Attributes:

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from `AscDescBase` .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

parent [(`ColorCorrection`)] The parent `ColorCorrection` instance that created this instance.

sat [(Decimal)] The saturation value (to be applied with Rec 709 coefficients) is stored here. Saturation is the last operation to be applied when applying a CDL.

sat can be set with a Decimal, float, int or numeric string.

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase` .

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

SopNode

Note: This class is meant only to be created by a `ColorCorrection` , and thus has the required arg of `parent` when instantiating it.

Warning: Setting any of the sop node values with a single value as in `offset = 5.4` will cause that value to be copied over all 3 colors, resulting in `[5.4, 5.4, 5.4]`.

class `cdl_convert.correction.`**SopNode**(*parent*)

Color node that contains slope, offset and power data.

Slope, offset and saturation are stored internally as lists, but always returned as tuples to prevent index assignment from being successful. This protects the user from inadvertently setting a single value in the list to be a non-valid value, which might result in values not being Decimals or even numbers at all.

Class Attributes:

element_names [[str]] Contains a list of XML Elements that refer to this class for use in parsing XML files.

Attributes:

desc [[str]] Since all Asc nodes which can contain a single description, can actually contain an infinite number of descriptions, the desc attribute is a list, allowing us to store every single description found during parsing.

Setting desc directly will cause the value given to append to the end of the list, but desc can also be replaced by passing it a list or tuple. Desc can be emptied by passing it None, [] or ().

Inherited from `AscDescBase` .

element [(<xml.etree.ElementTree.Element>)] etree style Element representing the node. Inherited from `AscXMLBase` .

parent [(`ColorCorrection`)] The parent `ColorCorrection` instance that created this instance.

slope [(Decimal, Decimal, Decimal)] An rgb tuple representing the slope, which changes the slope of the input without shifting the black level established by the offset. These values must be positive. If you set this attribute with a single value, it will be copied over all 3 colors. Any single value given can be a Decimal, float, int or numeric string.

default: (Decimal('1.0'), Decimal('1.0'), Decimal('1.0'))

offset [(Decimal, Decimal, Decimal)] An rgb tuple representing the offset, which raises or lowers the input brightness while holding the slope constant. If you set this attribute with a single value, it will be copied over all 3 colors. Any single value given can be a Decimal, float, int or numeric string.

default: (Decimal('0.0'), Decimal('0.0'), Decimal('0.0'))

power [(Decimal, Decimal, Decimal)] An rgb tuple representing the power, which is the only function that changes the response curve of the function. Note that this has the opposite response to adjustments than a traditional gamma operator. These values must be positive. If you set this attribute with a single value, it will be copied over all 3 colors. Any single value given can be a Decimal, float, int or numeric string.

default: (Decimal('1.0'), Decimal('1.0'), Decimal('1.0'))

xml [(str)] A nicely formatted XML string representing the node. Inherited from `AscXMLBase`.

xml_root [(str)] A nicely formatted XML, ready to write to file string representing the node. Formatted as an XML root, it includes the xml version and encoding tags on the first line. Inherited from `AscXMLBase`.

Public Methods:

build_element() Builds an ElementTree XML Element for this node and all nodes it contains. `element`, `xml`, and `xml_root` attributes use this to build the XML. This function is identical to calling the `element` attribute. Overrides inherited placeholder method from `AscXMLBase`.

parse_xml_descs() Parses an ElementTree Element for any Description tags and appends any text they contain to the `desc`. Inherited from `AscDescBase`

4.10.2 General Functions

Reset All

Resets all the class level lists and dictionaries of cdl_convert. Calling this is the same as calling each individual `reset_members` method.

`cdl_convert.`**`reset_all`**`()`
Resets all class level member lists and dictionaries

Sanity Check

`cdl_convert.utils.`**`sanity_check`**`(`*colcor*`)`
Checks values on `ColorCorrection` for sanity.

Args:

colcor [(`ColorCorrection`)] The `ColorCorrection` to check for sane values.

Returns:

(bool) Returns True if all values are sane.

Raises: N/A

Will print a warning to stdout if any values exceed normal limits. Normal limits are defined as:

For Slope, Power and Saturation: Any value over 3 or under 0.1

For Offset: Any value over 1 or under -1

Note that depending on the desired look for a shot or sequence, it's possible that a single ColorCorrection element might have very odd looking values and still achieve a correct look.

To Decimal

This is the function we use to convert ints, floats and strings to Decimal objects. We do NOT attempt to use maximum accuracy on floats passed in, as that results in extremely long values more often than not. Better to just truncate the float with a string conversion, than attempt to perfectly represent with a Decimal.

`cdl_convert.utils.`**`to_decimal`**`(`*value*, *name='Value'*`)`
Converts an incoming value to Decimal in the best way

Args:

value [(Decimal|str|float|int)] Any numeric value to be checked.

name='Value' [(str)] The type of value being checked: slope, offset, etc.

Returns:

(Decimal) If value passes all tests, returns value as Decimal.

Raises:

TypeError: If value given is not a number.

ValueError: If given a value that isn't an allowed type.

4.10.3 Parse Functions

These functions can either return a `ColorCorrection` , or a `ColorCollection` , depending on if they are from a container format.

Note: Use the `parse_file` function to parse any input file correctly, without worrying about matching the file extension by hand.

Parse ale

`cdl_convert.parse.`**`parse_ale`**(*input_file*)

Parses an Avid Log Exchange (ALE) file for CDLs

Args:

input_file [(str)] The filepath to the ALE EDL

Returns:

(`ColorCollection`) A collection that contains all found ColorCorrections

Raises: N/A

An ALE file is traditionally gathered during a telecine transfer using standard ASCII characters. Each line theoretically represents a single clip/take/shot.

Each field of data is tab delineated. We'll be searching for the ASC_SOP, ASC_SAT fields, alone with the standard Scan Filename fields.

The Data line indicates that all the following lines are comprised of shot information.

Parse cc

`cdl_convert.parse.`**`parse_cc`**(*input_file*)

Parses a .cc file for ASC CDL information

Args:

input_file [(str|<ElementTree.Element>)] The filepath to the CC or the `ElementTree.Element` object.

Returns:

(`ColorCorrection`) The `ColorCorrection` described within.

Raises:

ValueError: Bad XML formatting can raise ValueError is missing required elements.

A CC file is really only a single element of a larger CDL or CCC XML file, but this element has become a popular way of passing around single shot CDLs, rather than the much bulkier CDL file.

A sample CC XML file has text like:

```
    <ColorCorrection id="cc03340">
        <SOPNode>
            <Description>change +1 red, contrast boost</Description>
            <Slope>1.2 1.3 1.4</Slope>
            <Offset>0.3 0.0 0.0</Offset>
            <Power>1.0 1.0 1.0</Power>
        </SOPNode>
        <SatNode>
            <Saturation>1.2</Saturation>
        </SatNode>
    </ColorCorrection>
```

Additional elements can include multiple descriptions at every level, a description of the input colorspace, and a description of the viewing colorspace and equipment.

Parse ccc

cdl_convert.parse.**parse_ccc**(*input_file*)

Parses a .ccc file into a `ColorCollection` with type 'ccc'

Args:

input_file [(str)] The filepath to the CCC.

Returns:

(ColorCollection) A collection of all the found `ColorCorrection` as well as any metadata within the XML

Raises:

ValueError: Bad XML formatting can raise ValueError is missing required elements.

A ColorCorrectionCollection is just that- a collection of ColorCorrection elements. It does not contain any ColorDecision or MediaRef elements, but is free to contain as many Description elements as someone adds in.

It should also contain an InputDescription element, describing the color space and other properties of the incoming image, as well as a ViewingDescription which describes the viewing environment as well as any relevant hardware devices used to view or grade.

Parse cdl

cdl_convert.parse.**parse_cdl**(*input_file*)

Parses a .cdl file into a `ColorCollection` with type 'cdl'

Args:

input_file [(str)] The filepath to the CDL.

Returns:

(ColorCollection) A collection of all the found `ColorDecisions` as well as any metadata within the XML

Raises:

ValueError: Bad XML formatting can raise ValueError is missing required elements.

A ColorDecicionList is just that- a list of ColorDecision elements. It does not directly contain any ColorCorrections or Media Ref, only ColorDecisions, but is free to contain as many Description elements as someone adds in.

It should also contain an InputDescription element, describing the color space and other properties of the incoming image, as well as a ViewingDescription which describes the viewing environment as well as any relevant hardware devices used to view or grade.

Parse cmx

cdl_convert.parse.**parse_cmx**(*input_file*)

Parses a CMX EDL file for ASC CDL information.

Args:

input_file [(str)] The filepath to the CMX EDL

Returns:

(ColorCollection) A collection that contains all the ColorCorrection objects found within this EDL

Raises: N/A

001 DS0010.bg1 V C 00:08:07:23 00:08:16:10 01:00:00:00 01:00:08:11 *ASC_SOP (1.45 1.22 1.15)(-0.14 -0.11 -0.11)(1.00 1.00 1.00) *ASC_SAT 0.773000

Parse file

Passes on the file to the correct parser.

cdl_convert.parse.**parse_file**(*filepath*, *filetype=None*)

Determines & uses the correct parser to use on a CDL file

Args:

filepath [(str)] The filepath to the file. Must exist.

filetype=None [(str)] A file extension corresponding to the CDL type to convert from. If not provided, we'll derive it from the filepath.

Should not include a '.'

Raises: N/A

Returns:

ColorCorrection or ColorCollection Depending on the type of input file, this function will either return a single `ColorCorrection` or a full `ColorCollection` , containing one or more `ColorCorrection` or `ColorDecision`

Parse flex

cdl_convert.parse.**parse_flex**(*input_file*)

Parses a DaVinci FLEx telecine EDL for ASC CDL information.

Args:

input_file [(str)] The filepath to the FLEx EDL

Returns:

(ColorCollection) A collection that contains all the ColorCorrection objects found within this EDL

Raises: N/A

The DaVinci FLEx EDL is an odd duck, it's information conveyed via an extremely strict line & character addressing system.

Each line must begin with a line number header that indicated what type of information the line contains, with line number 100 indicating the start of a new shot/take. Lines 000-099 contain session information.

Within each line, important information is constricted to a certain range of characters, rather than space or comma separated like in an ALE EDL.

Some line numbers we care about, and the character indexes:

Line #	Line Name	Char Index	Data Type
010	Project Title	10-79	Title
100	Slate Info	10-17	Scene
		24-31	Take ID
		42-49	Camera Reel ID
701	ASC SOP	(This entry can be safely space separated)	
702	ASC SAT	(This entry can be safely space separated)	

We'll try and default to using the Slate information to derive the resultant filename, however that information is optional. If no slate information is found, we'll iterate up at the end of the title. If no title information is found, we'll have to iterate up on the actual input filename, which is far from ideal.

Parse Rhythm & Hues cdl

Rhythm & Hues' implementation of the cdl format is based on a very early spec, and as such lacks the all metadata. It's extremely unlikely that you'll run into this format in the wild.

`cdl_convert.parse.`**`parse_rnh_cdl`**(*input_file*)

Parses a space separated .cdl file for ASC CDL information.

Args:

input_file [(str)] The filepath to the CDL

Returns:

(`ColorCorrection`) The single ColorCorrection object retrieved from the beta CDL

Raises: N/A

A space separated cdl file is an internal Rhythm & Hues format used by the Rhythm & Hues for displaying shot level and sequence level within their internally developed playback software.

The file is a simple file consisting of one line. That line has 10, space separated elements that correspond to the ten ASC CDL elements in order of operations.

```
SlopeR SlopeG SlopeB OffsetR OffsetG OffsetB PowerR PowerG PowerB Sat
```

4.10.4 Write Functions

Each of these functions takes an `ColorCorrection` as an arg, then places as many attributes of the `ColorCorrection` that the format supports into a properly formatted string or XML Tree, then writes that file.

Write cc

`cdl_convert.write.`**`write_cc`**(*cdl*)

Writes the ColorCorrection to a .cc file

Write ccc

cdl_convert.write.**write_ccc**(*cdl*)
Writes the ColorCollection to a .ccc file

Write cdl

cdl_convert.write.**write_cdl**(*cdl*)
Writes the ColorCollection to a .cdl file

Write Rhythm & Hues cdl

This writes a very sparse cdl format that is based on a very early spec of the cdl implementation. It lacks all metadata. Unless you work at Rhythm & Hues, you probably don't want to write a cdl that uses this format.

cdl_convert.write.**write_rnh_cdl**(*cdl*)
Writes the ColorCorrection to a space separated .cdl file

CHAPTER 5

Indices and tables

- genindex
- search

A

AscColorSpaceBase (class in cdl_convert.base), 29
AscDescBase (class in cdl_convert.base), 30
AscXMLBase (class in cdl_convert.base), 30

C

ColorCollection (class in cdl_convert.collection), 31
ColorCorrection (class in cdl_convert.correction), 33
ColorCorrectionRef (class in cdl_convert.decision), 34
ColorDecision (class in cdl_convert.decision), 35
ColorNodeBase (class in cdl_convert.base), 37

M

MediaRef (class in cdl_convert.decision), 38

P

parse_ale() (in module cdl_convert.parse), 43
parse_cc() (in module cdl_convert.parse), 43
parse_ccc() (in module cdl_convert.parse), 44
parse_cdl() (in module cdl_convert.parse), 44
parse_cmx() (in module cdl_convert.parse), 45
parse_file() (in module cdl_convert.parse), 45
parse_flex() (in module cdl_convert.parse), 45
parse_rnh_cdl() (in module cdl_convert.parse), 46
Python Enhancement Proposals
    PEP 8, 24, 25, 27

R

reset_all() (in module cdl_convert), 42

S

sanity_check() (in module cdl_convert.utils), 42
SatNode (class in cdl_convert.correction), 40
SopNode (class in cdl_convert.correction), 41

T

to_decimal() (in module cdl_convert.utils), 42

W

write_cc() (in module cdl_convert.write), 46
write_ccc() (in module cdl_convert.write), 47
write_cdl() (in module cdl_convert.write), 47
write_rnh_cdl() (in module cdl_convert.write), 47