OpenColorIO Documentation

Contributors to the OpenColorIO Project

May 13, 2026

CONTENTS

1 About the Documentation **3**

2 Roadmap **5**

3 Community **7**
- 3.1 Meetings . . . 7
- 3.2 Slack . . . 7
- 3.3 Mailing Lists . . . 7

4 Related Projects **9**
- 4.1 ASWF Color Interop Forum . . . 9
- 4.2 NanoColor . . . 9

5 Search **11**
- 5.1 Quick Start . . . 11
  - 5.1.1 Quick Start for Artists . . . 11
  - 5.1.2 Quick Start for Config Authors . . . 11
  - 5.1.3 Quick Start for Developers . . . 12
  - 5.1.4 Quick Start for Contributors . . . 12
  - 5.1.5 Downloads . . . 12
  - 5.1.6 Installation . . . 13
  - 5.1.7 Installing OpenColorIO using Package Managers . . . 14
  - 5.1.8 Building from Source . . . 14
  - 5.1.9 Quick environment configuration . . . 20
  - 5.1.10 Environment variables . . . 21
- 5.2 Concepts . . . 21
  - 5.2.1 Overview . . . 21
  - 5.2.2 Introduction . . . 21
  - 5.2.3 Internal Architecture Overview . . . 24
  - 5.2.4 Glossary . . . 29
  - 5.2.5 Presentations & Publications . . . 29
- 5.3 Tutorials . . . 30
  - 5.3.1 Baking LUT's . . . 30
  - 5.3.2 Contributing . . . 36
- 5.4 Guides . . . 36
  - 5.4.1 Using OCIO . . . 36
  - 5.4.2 Environment Variables . . . 36
  - 5.4.3 Tool overview . . . 37
  - 5.4.4 Supported LUT Formats . . . 43
  - 5.4.5 Compatible Software . . . 44

5.4.6 FAQ . . . 53
5.4.7 Authoring Configurations . . . 54
5.4.8 Config syntax . . . 54
5.4.9 Looks Example . . . 85
5.4.10 Contexts Example . . . 87
5.4.11 How to Configure ColorSpace Allocation . . . 93
5.4.12 Developing with OCIO . . . 94
5.4.13 Usage Examples . . . 94
5.4.14 Application Helpers . . . 101
5.4.15 Contributing . . . 102
5.4.16 Getting started . . . 102
5.4.17 Repository structure . . . 104
5.4.18 Architectural notes . . . 106
5.4.19 Coding style guide . . . 106
5.4.20 Unit tests . . . 108
5.4.21 Test framework . . . 109
5.4.22 Issues . . . 109
5.4.23 Submitting an issue . . . 110
5.4.24 Submitting Changes . . . 110
5.4.25 Documentation guidelines . . . 111
5.4.26 OpenColorIO Doxygen Style Guide . . . 114
5.5 Configurations . . . 116
5.5.1 OCIO Studio Config for ACES . . . 116
5.5.2 OCIO CG Config for ACES . . . 117
5.5.3 ocio-v2_demo . . . 118
5.5.4 aces_1.0.3 . . . 131
5.5.5 spi-anim . . . 139
5.5.6 spi-vfx . . . 139
5.5.7 nuke-default . . . 146
5.6 Releases . . . 147
5.6.1 OCIO 2.5 Release . . . 147
5.6.2 New Feature Guide . . . 147
5.6.3 Release Notes . . . 152
5.6.4 OCIO 2.4 Release . . . 152
5.6.5 New Feature Guide . . . 152
5.6.6 Release Notes . . . 156
5.6.7 OCIO 2.3 Release . . . 156
5.6.8 New Feature Guide . . . 156
5.6.9 Release Notes . . . 159
5.6.10 OCIO 2.2 Release . . . 160
5.6.11 New Feature Guide . . . 160
5.6.12 Release Notes . . . 165
5.6.13 OCIO 2.1 Release . . . 166
5.6.14 New Feature Guide . . . 166
5.6.15 Release Notes . . . 168
5.6.16 OCIO 2.0 Release . . . 168
5.6.17 New Feature Guide . . . 169
5.6.18 Changes from v1 . . . 175
5.6.19 Release Notes . . . 177
5.7 API . . . 177
5.7.1 Applications Helpers . . . 177
5.7.2 Baker . . . 194
5.7.3 ColorSpace . . . 198
5.7.4 ColorSpaceSet . . . 206

5.7.5 Config . . . 210
5.7.6 Constants . . . 253
5.7.7 Context . . . 256
5.7.8 DynamicProperty . . . 261
5.7.9 DynamicPropertyValue . . . 264
5.7.10 DynamicPropertyDouble . . . 265
5.7.11 DynamicPropertyGradingPrimary . . . 265
5.7.12 DynamicPropertyGradingRGBCurve . . . 266
5.7.13 DynamicPropertyGradingTone . . . 267
5.7.14 Enums . . . 267
5.7.15 Exceptions . . . 299
5.7.16 FormatMetadata . . . 300
5.7.17 Global . . . 303
5.7.18 Grading Transforms . . . 311
5.7.19 ImageDesc . . . 334
5.7.20 Look . . . 341
5.7.21 Named Transform . . . 343
5.7.22 Processors . . . 347
5.7.23 Rules . . . 357
5.7.24 Shaders . . . 366
5.7.25 SystemMonitors . . . 387
5.7.26 Transforms . . . 388
5.7.27 ViewTransform . . . 437
5.8 ASWF . . . 441
5.8.1 License . . . 441
5.8.2 Charter . . . 441
5.8.3 CLA and DCO . . . 444
5.8.4 ASWF Docker . . . 446

Index **447**

OpenColorIO (OCIO) is a complete color management solution geared towards motion picture production with an emphasis on visual effects and computer animation. OCIO provides a straightforward and consistent user experience across all supporting applications while allowing for sophisticated back-end configuration options suitable for high-end production usage. OCIO is compatible with the Academy Color Encoding Specification (ACES) and is LUT-format agnostic, supporting many popular formats.

OpenColorIO version 1.0 was released in 2011, having been in development since 2003. OCIO v1 represented the culmination of years of production experience earned on such films as SpiderMan 2 (2004), Surf's Up (2007), Cloudy with a Chance of Meatballs (2009), Alice in Wonderland (2010), and many more.

OpenColorIO v2 was in development from 2017 to 2020 and was feature complete as of SIGGRAPH 2020 and was released in January of 2021. Development has continued to be active since then with annual releases around September, timed in relation to the VFX Reference Platform. Here is more information about our *Releases*.

CHAPTER

ONE

ABOUT THE DOCUMENTATION

The documentation has been updated to include basic coverage of the new OCIO v2 features but additional work is needed to provide more detail and tutorials about how to best leverage them. We have an OCIO User Experience (UX) working group that is gradually working on more coverage.

The documentation is a work-in-progress and we would love to have your help to improve it! An easy way to get involved is to join the #opencolorio channel on *Slack*.

CHAPTER

TWO

ROADMAP

The project attempts to maintain a roadmap to give the community insight into upcoming development. The items are grouped into “Now”, “Next”, and “Later” categories. “Now” indicates work that is currently in-progress. “Next” indicates work that is being planned for imminent development (meaning now would be a good time to provide your input). The roadmap is available here.

CHAPTER

THREE

COMMUNITY

3.1 Meetings

The OpenColorIO project has a Technical Steering Committee meeting on Zoom every two weeks at noon LA time. The Zoom link is available from the OCIO Calendar.

There is a general working group and "office hours" meeting once a month in the same time slot. The Zoom link is available from the OCIO Calendar.

These meetings are open to anyone!

3.2 Slack

Join us on the ASWF Slack in the channels *#opencolorio* and *#opencolor-configs*.

3.3 Mailing Lists

Most of the conversation happens on Slack, but there are two mailing lists associated with OpenColorIO:

ocio-user@lists.aswf.io
: For end users (artists, often) interested in OCIO profile design, facility color management, and workflow.

ocio-dev@lists.aswf.io
: For developers interested OCIO APIs, code integration, compilation, etc.

CHAPTER

FOUR

RELATED PROJECTS

4.1 ASWF Color Interop Forum

The CIF is an open, cross-project forum for discussing color interoperability across varying workflows and standards. The aim isn't necessarily to develop new standards, but rather to make recommendations for how to improve color accuracy through the existing infrastructure.

Recommendations are published on the ColorInterop GitHub.

The forum has a monthly Zoom meeting on Mondays at noon LA time, as posted on the OCIO Calendar.

Discussion happens in the *#color-interop-forum* channel on the ASWF Slack.

4.2 NanoColor

NanoColor is a collaboration between OpenUSD, MaterialX, and OpenColorIO to ensure that there is interoperable and flexible color management among those projects and related projects in the computer graphics world.

The working group has a Zoom meeting every two weeks on Mondays at 1:00 pm LA time, as posted on the OCIO Calendar.

Discussion happens in the *#nanocolor* channel on the ASWF Slack.

CHAPTER

FIVE

SEARCH

search

genindex

5.1 Quick Start

5.1.1 Quick Start for Artists

Most users will likely want to use the OpenColorIO that comes precompiled with their applications. See the *Compatible Software* for further details on each application.

Note that OCIO configurations are required to do any 'real' work, and are available separately on the *Downloads* section of this site. Example images are also available. For assistance customizing .ocio configurations, contact ocio-user or the #configs channel on *Slack*

If your application supports OCIO 2.2 or later, you may take advantage of the configurations built into OCIO itself. For example, by setting the config path to: `ocio://default`.

- Step 1: set the `$OCIO` environment-variable to `/path/to/your/config.ocio`
- Step 2: Launch supported application.

If you are on a platform that is not envvar friendly, most applications also provide a menu option to select a different OCIO configuration after launch.

Please be sure to select a configuration that matches your color workflow (VFX work sometimes requires a different profile than animated features). If you need assistance picking a profile, email ocio-user.

5.1.2 Quick Start for Config Authors

As a config author, you'll want access to the OCIO command-line tools such as `ociocheck`. Please see the *Installation* instructions for the various options, for example Homebrew on macOS and Vcpkg on Windows. On Linux you may need to build from source.

Grab the available configuration files (and the sample images, if you want) from *Downloads* so you'll have some examples to study.

Try using the command-line tool *ociochecklut* to send an RGB through a file in the "luts" sub-directory of one of the configs you downloaded. Try using the *ocioconvert* tool to process an image from one color space to another. Try using *ociodisplay* to view an image through a color space conversion.

Look through the *Overview* to get a sense of the big picture.

Study the descriptions in the *Configurations* section of the example configs.

Check out *Releases* for an overview of the new features in OCIO v2.

5.1.3 Quick Start for Developers

Get started by following the *Installation* instructions to build OCIO from source. In addition to the library itself, you will want to make sure to build the command-line tools, including `ocioconvert`.

Note that active development of OCIO v2 is happening in the main branch.

Watch the OCIO SIGGRAPH course listed in *Presentations & Publications*.

Grab the available configuration files (and the sample images, if you want) from *Downloads*.

Try using the command-line tool *ociochecklut* to send an RGB through a file in the "luts" sub-directory of one of the configs you downloaded. Try using the *ocioconvert* tool to process an image from one color space to another. Try using *ociodisplay* to view an image through a color space conversion.

Take a look at the example code in the developers-usageexamples and then study the code for the command-line tools, which is under `src/apps`.

There are helper classes under `src/libutils/apphelpers` for common application tasks such as generating color space menus, building a viewing pipeline for a viewport, and writing a color picker for scene-linear color spaces.

Look through the *Overview* to get a sense of the big picture.

Check out *Releases* for an overview of the new features in OCIO v2.

Join the #dev and #ux channel on *Slack*. From the #ux channel, you may want to get involved with the OCIO User Experience (UX) working group which discusses best practices for implementing OCIO support in applications.

5.1.4 Quick Start for Contributors

First decide how you want to contribute to OCIO. We're looking for help with testing, example configs, and documentation as well as actual coding.

- If you want to write code, start by looking at the *Quick Start for Developers* and join the #dev channel on *Slack*. Take a look on GitHub at the issues with the `good first issue` label.
- If you want to help develop config files, start by looking at *Quick Start for Config Authors* and join the #configs channel on *Slack*. From there, you may want to join the OCIO Configs working group.
- If you want to help with documentation, join the #docs and #ux channel on *Slack*.
- If you want to help test, join the #general channel on *Slack* and let us know you want to help test.

The *Contributing* guide has a ton of useful information about most aspects of contributing to OCIO.

Check out *Releases* for an overview of the new features in OCIO v2.

5.1.5 Downloads

- OCIO v2 ACES Configurations
- Reference Images v1.0v4 – .tgz
- OCIO Library source code

Contributor License Agreements

Please see the OCIO GitHub repository

5.1.6 Installation

Installation may be done either by installing pre-built binaries from various package managers or by building from source. (The OCIO project currently does not provide pre-built libraries as downloadable artifacts other than through package managers.)

Please note that the version available through a given package manager may be significantly outdated when compared to the current official OCIO release, so please verify the version you install is current. (Unfortunately, even two years after the introduction of OpenColorIO v2, several package managers continue to install OCIO 1.1.1.)

Alternatives

Python

If you only need the Python binding and command-line tools, the simplest solution is to take advantage of the pre-built wheels in the Python Package Index (PyPI) here. It can be installed as follows, once you have Python installed.

PyPI:

```
pip install opencolorio
```

The pre-built wheels are listed here. Note that source code is provided, so it may be possible for `pip` to compile the binding on your machine if the matrix of Python version and platform version does not have the combination you need. More detailed instructions are available for how to use pip.

OpenImageIO

If you only need to apply color conversions to images, please note that OpenImageIO's `oiiotool` has most of the functionality of the `ocioconvert` command-line tool (although not everything, such as GPU processing). OpenImageIO is available via several package managers (including Homebrew and Vcpkg).

Homebrew:

```
brew install openimageio
```

Vcpkg:

```
vcpkg install openimageio[opencolorio,tools]:x64-windows --recurse
```

5.1.7 Installing OpenColorIO using Package Managers

Linux

When it comes to Linux distributions, relatively few of the Linux distribution repositories have been updated to OCIO v2. The **latest Fedora** is one good option as it offers a recent release of OpenColorIO v2. Information about the package can be found on the Fedora project website.

For the other distributions, information about which release of OpenColorIO is available can be verified on pkgs.org.

The recommendation is to build OpenColorIO from source. You may build from source using the instructions below. See *Building from Source*.

Windows using Vcpkg

Vcpkg can be used to install OpenColorIO on Windows 7 or higher. To do that, Vcpkg must be installed by following the official instructions. Once Vcpkg is installed, OpenColorIO and some of the tools can be installed with the following command:

```
vcpkg install opencolorio[tools]:x64-windows
```

Note that this package **does not** install `ocioconvert`, `ociodisplay`, `ociolutimage`, and the Python binding.

If you need the extra command-line tools, you'll need to install *from source.* However, the Python binding can be installed as described in the *Python* section.

MacOS using Homebrew

You can use the Homebrew package manager to install OpenColorIO on macOS.

First install Homebrew as per the instructions on the Homebrew homepage (or see the Homebrew documentation for more detailed instructions).

Then simply run the following command to install:

```
brew install opencolorio
```

Homebrew does not install the Python binding or the command-line tools that depend on either OpenImageIO or OpenEXR such as `ocioconvert`, `ociodisplay`, and `ociolutimage`.

5.1.8 Building from Source

The basic requirements to build OCIO from source on Linux, macOS, and Windows are a supported C++ compiler, CMake, and an internet connection. The OCIO Cmake scripts are able to download, build, and install all other required components. However, more advanced users are also able to have the build use their own version of dependencies.

Dependencies

OCIO depends on the following components. By default, the OCIO CMake scripts will download and build the items labelled with a *, so it is not necessary to install those items manually:

Required components:

- C++ 17-23 compiler (gcc, clang, msvc)
- CMake >= 3.14
- *Expat >= 2.6.0 (XML parser for CDL/CLF/CTF)
- *yaml-cpp >= 0.8.0 (YAML parser for Configs)
- *Imath >= 3.1.1 (for half domain LUTs)
- *pystring >= 1.1.3
- *minizip-ng >= 4.0.0 (for config archiving)
- *ZLIB >= 1.2.13 (for config archiving)

Optional OCIO functionality also depends on:

- *Little CMS >= 2.2 (for ociobakelut ICC profile baking)
- *OpenGL GLUT & GLEW (for ociodisplay)
- *OpenEXR >= 3.2.0 (for apps including ocioconvert)
- OpenImageIO >= 2.2.14 (for apps including ocioconvert)
- *OpenFX >= 1.4 (for the OpenFX plug-ins)
- OpenShadingLanguage >= 1.13 (for the OSL unit tests)
- Doxygen (for the docs)
- NumPy (optionally used in the Python test suite)
- *pybind11 >= 2.9.2 (for the Python binding)
- Python >= 3.9 (for the Python binding only)
- Python 3.9+ (for building the documentation)

Building the documentation requires the following packages, available via PyPI:

- Sphinx
- six
- testresources
- recommonmark
- sphinx-press-theme
- sphinx-tabs
- breathe

Example bash scripts are provided in share/ci/scripts for installing some dependencies. These are used by OpenColorIO CI so are regularly tested on their noted platforms. The `install_docs_env.sh` script will install all dependencies for building OCIO documentation and is available for all supported platforms. Use GitBash (provided with Git for Windows) to execute this script on Windows.

Automated Installation

Dependencies listed above with a preceeding * can be automatically installed at build time using the OCIO_INSTALL_EXT_PACKAGES option in your cmake command (requires an internet connection). The C/C++ libraries are pulled from external repositories, built, and are (typically) statically-linked into an OCIO dynamic library. All installs of these components are fully contained within your build directory.

Three OCIO_INSTALL_EXT_PACKAGES options are available:

```
cmake -DOCIO_INSTALL_EXT_PACKAGES=<NONE|MISSING|ALL>
```

- NONE: Use system installed packages. Fail if any are missing or don't meet minimum version requirements.
- MISSING (default): Prefer system installed packages. Install any that are not found or don't meet minimum version requirements.
- ALL: Install all required packages, regardless of availability on the current system.

Existing Install Hints

If the library is not installed in a typical location where CMake will find it, you may specify the location using one of the following methods:

- Set -D<package_name>_DIR to point to the directory containing the CMake configuration file for the package.
- Set -D<package_name>_ROOT to point to the directory containing the lib and include directories.
- Set -D<package_name>_LIBRARY and -D<package_name>_INCLUDE_DIR to point to the lib and include directories.

Not all packages support all of the above options. Please refer the OCIO CMake find modules for the package that you are having trouble with to see the options it supports.

Usually CMake will use the dynamic library rather than static, if both are present. In this case, you may set <package_name>_STATIC_LIBRARY to ON to request use of the static one. If only the static library is present (such as when OCIO builds the dependency), then the option is not needed. The following packages support this option: expat, yaml-cpp, Imath, lcms2, and minizip-ng. Using CMake 3.24+, it is possible to prefer the static version of ZLIB with -DZLIB_USE_STATIC_LIBS=ON.

The package names used by OCIO are as follows (note that these are case-sensitive):

Required:

- expat
- yaml-cpp
- Imath
- pystring
- ZLIB
- minizip-ng

Optional:

- OpenEXR
- OpenImageIO
- lcms2

- pybind11
- openfx
- OSL
- Sphinx
- GLEW
- GLUT
- Python

Please note that if you provide your own minizip-ng, rather than having OCIO's CMake download and build it, you will likely need to set its CMake variables the same way that OCIO does (e.g., enable ZLib and turn off most other options). Using a minizip-ng from various package managers (e.g., Homebrew) probably won't work. Please see the settings that begin with -DMZ_ that are used in the OCIO minizip-ng find module.

Please note that if you build a static OCIO library, it will not contain the libraries for the external packages and so you will need to list those separately when linking your client application. If you had OCIO download and build these packages, you will find them under your build directory in ext/dist/lib. The libraries that are needed are: expat, yaml-cpp, Imath, pystring, ZLIB, and minizip-ng.

The OCIO make install step will install CMake configuration files that may be used by applications that consume OCIO to find and utilize OCIO during their own build process.

For the Python packages required for the documentation, ensure that their locations are in your PYTHONPATH environment variable prior to configuring the build.

This custom variable is also available:

- -DPython_EXECUTABLE=<path> (Python executable for pybind11)

Enabling optional components

CMake Options

There are many options available in `CMake.

<https://cmake.org/cmake/help/latest/guide/user-interaction/index.html#guide:User%20Interaction%20Guide>`_

Several of the most common ones are:

- -DCMAKE_BUILD_TYPE=Release (Set to Debug, if necessary)
- -DBUILD_SHARED_LIBS=ON (Set to OFF to build OCIO as a static library)

Here are the most common OCIO-specific CMake options (the default values are shown):

- -DOCIO_BUILD_APPS=ON (Set to OFF to not build command-line tools)
- -DOCIO_USE_OIIO_FOR_APPS=OFF (Set ON to build tools with OpenImageIO rather than OpenEXR)
- -DOCIO_BUILD_PYTHON=ON (Set to OFF to not build the Python binding)
- -DOCIO_BUILD_OPENFX=OFF (Set to ON to build the OpenFX plug-ins)
- -DOCIO_USE_SIMD=ON (Set to OFF to turn off SIMD CPU performance optimizations, such as SSE and NEON)
- -DOCIO_USE_SSE2 (Set to OFF to turn off SSE2 CPU performance optimizations)
- -DOCIO_USE_AVX (Set to OFF to turn off AVX CPU performance optimizations)
- -DOCIO_USE_AVX2 (Set to OFF to turn off AVX2 CPU performance optimizations)

- -DOCIO_USE_F16C (Set to OFF to turn off F16C CPU performance optimizations)
- -DOCIO_BUILD_TESTS=ON (Set to OFF to not build the unit tests)
- -DOCIO_BUILD_GPU_TESTS=ON (Set to OFF to not build the GPU unit tests)
- -DOCIO_USE_HEADLESS=OFF (Set to ON to do headless GPU rendering)
- -DOCIO_WARNING_AS_ERROR=ON (Set to OFF to turn off warnings as errors)
- -DOCIO_BUILD_DOCS=OFF (Set to ON to build the documentation)

Note that OCIO will turn off any specific SIMD CPU performance optimizations if they are not supported by the build target architecture. The default for OCIO_USE_SSE2, OCIO_USE_AVX, OCIO_USE_AVX2 and OCIO_USE_F16C depends on the architecture, but will be ON where supported.

On MacOS, the default is to build for the native architecture that CMake is running under. For example, if a x86_64 version of CMake is running under Rosetta, the native architecture will be x86_64, rather then arm64. You can use the CMAKE_OSX_ARCHITECTURES option to override that. To build universal binaries, use the following option: -DCMAKE_OSX_ARCHITECTURES="arm64;x86_64".

When doing a universal build, note that the OCIO dependencies must be built as universal libraries too. If you are running in OCIO_INSTALL_EXT_PACKAGES=MISSING or NONE mode, your build will fail if any of your installed libraries are not universal. The easiest way to address this is to set OCIO_INSTALL_EXT_PACKAGES=ALL in order to let OCIO build everything. Alternatively, you may set CMAKE_OSX_ARCHITECTURES to just the platform you are targeting.

Several command-line tools (such as ocioconvert) require reading or writing image files. If OCIO_USE_OIIO_FOR_APPS=OFF, these will be built using OpenEXR rather than OpenImageIO and therefore you will be limited to using OpenEXR files with these tools rather than the wider range of image file formats supported by OIIO. (Using OpenEXR for these tools works around the issue of a circular dependency between OCIO and OIIO that can complicate some build chains.)

The CMake output prints information regarding which image library will be used for the command-line tools (as well as a lot of other info about the build configuration).

Documentation

Instructions for installing the documentation pre-requisites and building the docs are in the section on contributing documentation.

MacOS and Linux

While there is a huge range of possible setups, the following steps should work on macOS and most Linux distros. To keep things simple, this guide will use the following example paths - these will almost definitely be different for you:

- source code: /source/ocio
- the temporary build location: /tmp/ociobuild
- the final install directory: /software/ocio

First make the build directory and cd to it:

```
$ mkdir /tmp/ociobuild
$ cd /tmp/ociobuild
```

Next step is to run cmake, which looks for things such as the compiler's required arguments, optional requirements like Python, OpenImageIO etc

For this example we will show how to install OCIO to a custom location (instead of the default `/usr/local`), we will thus run `cmake` with `CMAKE_INSTALL_PREFIX`.

Still in `/tmp/ociobuild`, run:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/software/ocio /source/ocio
```

The last argument is the location of the OCIO source code (containing the main CMakeLists.txt file). You should see it conclude with something along the lines of:

```
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/ociobuild
```

Next, build everything (with the `-j` flag to build using 8 threads):

```
$ make -j8
```

Starting with CMake 3.12, you can instead run a portable parallel build:

```
$ cmake --build . -j 8
```

This should complete in a few minutes. Finally, install the files into the specified location:

```
$ make install
```

If nothing went wrong, `/software/ocio` should look something like this (on Linux or macOS):

```
$ cd /software/ocio
$ ls
bin/  include/  lib/  share/
$ ls bin/
ociobakelut ociocheck  (and others ...)
$ ls include/
OpenColorIO/
$ ls lib/
cmake/  libOpenColorIO.dylib  (and some more specific versions ...)
libOpenColorIOimageioapphelpers.a  libOpenColorIOoglapphelpers.a
pkgconfig/  python<version>/
$ ls lib/pkgconfig
OpenColorIO.pc
$ ls lib/python<version>/site-packages
PyOpenColorIO.so
$ ls share/ocio
setup_ocio.sh
```

Windows

While build environments may vary between users, the recommended way to build OCIO from source on Windows 7 or newer is to use the scripts provided in the Windows share section of the OCIO repository. There are two scripts currently available.

The first script is called ocio_deps.bat and it provides some automation to install the most difficult dependencies. Those dependencies are:

- Vcpkg
- OpenImageIO
- Freeglut
- Glew
- Python dependencies for documentation

Run this command to execute the ocio_deps.bat script:

```
ocio_deps.bat --vcpkg <path to current vcpkg installation or where it should be␣
↪installed>
```

The second script is called ocio.bat and it provide a way to configure and build OCIO from source. Moreover, this script executes the install step of `cmake` as well as the unit tests. The main use case is the following:

```
ocio.bat --b <path to build folder> --i <path to install folder>
--vcpkg <path to vcpkg installation> --ocio <path to ocio repository> --type Release
```

For more information, please look at each script's documentation:

```
ocio.bat --help

ocio_deps.bat --help
```

5.1.9 Quick environment configuration

The quickest way to set the required *Environment variables* is to source the `share/ocio/setup_ocio.sh` script installed with OCIO. On Windows, use the corresponding setup_ocio.bat file. See OCIO's install directory under share/ocio.

For a temporary configuration of your terminal, you can run the following script:

```
# Windows - Execute setup_ocio.bat
[... path to OCIO install directory]/share/ocio/setup_ocio.bat
# Unix - Execute setup_ocio.sh
[... path to OCIO install directory]\share\ocio\setup_ocio.sh
```

For a more permanent option, add the following to `~/.bashrc` (assuming you are using bash, and the example install directory of `/software/ocio`):

```
source /software/ocio/share/ocio/setup_ocio.sh
```

The only environment variable you must configure manually is *OCIO*, which points to the configuration file you wish to use. For prebuilt config files, see the *Downloads* section

To do this, you would add a line to `~/.bashrc` (or a per-project configuration script etc), for example:

```
export OCIO="/path/to/my/config.ocio"
```

5.1.10 Environment variables

Note: For other user facing environment variables, see *Environment Variables*.

OCIO

This variable needs to point to the global OCIO config file, e.g `config.ocio`

DYLD_LIBRARY_PATH (macOS)

The `lib/` folder (containing `libOpenColorIO.dylib`) must be on the `DYLD_LIBRARY_PATH` search path, or you will get an error similar to:

```
dlopen(.../OCIOColorSpace.so, 2): Library not loaded: libOpenColorIO.dylib
Referenced from: .../OCIOColorSpace.so
Reason: image not found
```

This applies to anything that links against OCIO, including the `PyOpenColorIO` Python binding.

LD_LIBRARY_PATH (Linux)

The Linux equivalent of the macOS `DYLD_LIBRARY_PATH`.

PYTHONPATH

Python's module search path. If you are using the PyOpenColorIO module, you must add `lib/python2.x` to this search path (e.g `python/2.5`), or importing the module will fail:

```
>>> import PyOpenColorIO
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named PyOpenColorIO
```

Note that `DYLD_LIBRARY_PATH` or `LD_LIBRARY_PATH` must be set correctly for the module to work.

OFX_PLUGIN_PATH

When building the OCIO OpenFX plugins, include the installed `OpenColorIO/lib` directory (where `OpenColorIO.ofx.bundle` is located) in this path.

It is recommended to build OFX plugins in static mode (`BUILD_SHARED_LIBS=OFF`) to avoid any issue loading the OpenColorIO library from the plugin once it has been moved. Otherwise, please make sure the shared OpenColorIO lib (*.so, *.dll, *.dylib) is visible from the plugin by mean of `PATH`, `LD_LIBRARY_PATH` or `DYLD_LIBRARY_PATH` for Windows, Linux and macOS respectively. For systems that supports it, it is also possible to edit the RPATH of the plugin to add the location of the shared OpenColorIO lib.

5.2 Concepts

5.2.1 Overview

5.2.2 Introduction

Warning: This section is from OCIO v1 and has not been updated yet.

OpenColorIO (OCIO) is a complete color management solution geared towards motion picture production with an emphasis on visual effects and computer animation. As such, OCIO helps enforce a color management methodology that is required for the high fidelity color imaging in modern computer graphics. This section introduces those concepts and general workflow practices. Additional information can be found in Jeremy Selan's Cinematic Color document.

While OCIO is a color management library, its only knowledge of color science comes from its execution of the transforms defined in the OCIO configuration file. These transforms are either defined by the end user in a custom OCIO config or inherited from the publicly available configs.

By specifying your desired `config.ocio` Config file in the local environment all OCIO compatible applications and software libraries will be able to see your defined color transform "universe", and direct the transformation of image data from one defined `OCIO.ColorSpace` to another, in addition to the other transforms documented elsewhere.

Sony Pictures Imageworks Color Pipeline

This document describes a high-level overview on how to emulate the current color management practice at Sony Imageworks. It applies equally to all profiles used at Imageworks, including both the VFX and Animation profiles. It's by no means a requirement to follow this workflow at your own facility, this is merely a guideline on how we choose to work.

General Pipeline Observations

- All images, on disk, contain colorspace information as a substring within the filename. This is obeyed by all applications that load image, write images, or view images. File extensions and metadata are ignored with regards to color processing.

Example:

```
colorimage_lnf.exr  : lnf
dataimage_ncf.exr : ncf
plate_lg10.dpx : lg10
texture_dt8.tif : dt8
```

Note: File format extension does **NOT** imply a color space. Not all .dpx files are lg10. Not all .tif images are dt8.

- The common file formats we use are exr, tif, dpx.
- render outputs: exr
- render inputs (mipmapped-textures): exr, tif (txtif)
- photographic plates (scans): dpx
- composite outputs: dpx, exr
- on-set reference: (camera raw) NEF, CR2, etc.
- painted textures: psd, tif
- output proxies: jpg
- All pipelines that need to be colorspace aware rely on `Config.parseColorSpaceFromString`.
- Color configurations are show specific. The `$OCIO` environment variable is set as part of a 'setshot' process, before other applications are launched. Artists are not allowed to work across different shows without using a fresh shell + setshot.

- While the list of colorspaces can be show specific, care is taken to maintain similar naming to the greatest extent feasible. This reduces artist confusion. Even if two color spaces are not identical across shows, if they serve a similar purpose they are named the same.
- Example: We label 10-bit scanned film negatives as lg10. Even if two different shows use different acquisition film stocks, and rely on different linearization curves, they are both labeled lg10.
- There is no explicit guarantee that image assets copied across shows will be transferable in a color-correct manner. For example, in the above film scan example, one would not expect that the linearized versions of scans processed on different shows to match. In practice, this is not a problematic issue as the colorspaces which are convenient to copy (such as texture assets) happen to be similarly defined across show profiles.

Rendering

- Rendering and shading occurs in a scene-linear floating point space, typically named "ln". Half-float (16-bit) images are labeled lnh, full float images (32-bit) are labeled lnf.
- All image inputs should be converted to ln prior to render-time. Typically, this is done when textures are published. (See below)
- Renderer outputs are always floating-point. Color outputs are typically stored as lnh (16-bit half float).
- Data outputs (normals, depth data, etc) are stored as ncf ("not color" data, 32-bit full float). Lossy compression is never utilized.
- Render outputs are always viewed with an OCIO compatible image viewer. Thus, for typical color imagery the lnf display transform will be applied. In Nuke, this can be emulated using the *OCIODisplay* node. A standalone image viewer, ociodisplay, is also included with OpenColorIO src/example.

Texture Painting / Matte Painting

- Textures are painted either in a non-OCIO color-managed environment (Photoshop, etc), or a color managed one like Mari.
- At texture publish time, before mipmaps are generated, all color processing is applied. Internally at SPI we use OpenImageIO's maketx that also links to OpenColorIO. This code is available on the public OIIO repository. Color processing (linearization) is applied before mipmap generation in order to assure energy preservation in the render. If the opposite processing order were used, (mipmap in the original space, color convert in the shader), the apparent intensity of texture values would change as the object approached or receded from the camera.
- The original texture filenames contain the colorspace information as a substring, to signify processing intent.
- Textures that contain data (bump maps, opacity maps, blend maps, etc) are labeled with the nc colorspaces according to their bitdepth.
- Example: an 8-bit opacity map -> skin_opacity_nc8.tif
- Painted textures that are intended to modulate diffuse color components are labeled dt (standing for "diffuse texture"). The dt8 colorspace is designed such that, when linearized, values will not extend above 1.0. At texture publishing time these are converted to lnh mipmapped tiffs/exr. Note that as linear textures have greater allocation requirements, a bit depth promotion is required in this case. I.e., even if the original texture as painted was only 8-bits, the mipmapped texture will be stored as a 16-bit float image.
- Painted environment maps, which may be emissive as labeled vd (standing for 'video'). These values, when linearized, have the potential to generate specular information well above 1.0. Note that in the current vd linearization curves, the top code values may be very "sensitive". I.e., very small changes in the initial code value

(such as 254->255) may actually result in very large differences in the estimated scene-linear intensity. All environment maps are store as lnh mipmapped tiffs/exr. The same bit-depth promotion as in the dt8 case is required here.

Compositing

- The majority of compositing operations happen in scene-linear, lnf, colorspace.
- All image inputs are linearized to lnf as they are loaded. Customized input nodes make this processing convenient. Rendered elements, which are stored in linear already, do not require processing. Photographic plates will typically be linearized according to their source type, (lg10 for film scans, gn10 for genesis sources, etc).
- All output images are de-linearized from lnf when they are written. A customized output node makes this convenient.
- On occasion log data is required for certain processing operations. (Plate resizing, pulling keys, degrain, etc). For each show, a colorspace is specified as appropriate for this operation. The artist does not have to keep track of which colorspace is appropriate to use; the OCIOLogConvert node is always intended for this purpose. (Within the OCIO profile, this is specified using the 'compositing_log' role).

Further Information

Specific information with regard to the public OCIO configs can be found in the *Configurations* section.

5.2.3 Internal Architecture Overview

Warning: This section is from OCIO v1 and has not been updated yet.

External API

Configs

At the highest level, we have OCIO::Configs. This represents the entirety of the current color "universe". Configs are serialized as .ocio files, read at runtime, and are often used in a 'read-only' context.

Config are loaded at runtime to allow for customized color handling in a show- dependent manner.

Example Configs:

- ACES (Academy's standard color workflow)
- spi-vfx (Used on some Imageworks VFX shows such as spiderman, etc).
- and others

ColorSpaces

The meat of an OCIO::Config is a list of named ColorSpaces. ColorSpace often correspond to input image states, output image states, or image states used for internal processing.

Example ColorSpaces (from ACES configuration):

- aces (HDR, scene-linear)
- adx10 (log-like density encoding space)
- slogf35 (sony F35 slog camera encoding)
- rrt_srgb (baked in display transform, suitable for srgb display)
- rrt_p3dci (baked in display transform, suitable for dcip3 display)

Transforms

ColorSpaces contain an ordered list of transforms, which define the conversion to and from the Config's "reference" space.

Transforms are the atomic units available to the designer in order to specify a color conversion.

Examples of OCIO::Transforms are:

- File-based transforms (1d lut, 3d lut, mtx… anything, really.)
- Math functions (gamma, log, mtx)
- The 'meta' GroupTransform, which contains itself an ordered lists of transforms
- The 'meta' LookTransform, which contains an ordered lists of transforms

For example, the adx10 ColorSpace (in one particular ACES configuration) -Transform FROM adx, to our reference ColorSpace:

1. Apply FileTransform adx_adx10_to_cdd.spimtx
2. Apply FileTransform adx_cdd_to_cid.spimtx
3. Apply FileTransform adx_cid_to_rle.spi1d
4. Apply LogTransform base 10 (inverse)
5. Apply FileTransform adx_exp_to_aces.spimtx

If we have an image in the reference ColorSpace (unnamed), we can convert TO adx by applying each in the inverse direction:

1. Apply FileTransform adx_exp_to_aces.spimtx (inverse)
2. Apply LogTransform base 10 (forward)
3. Apply FileTransform adx_cid_to_rle.spi1d (inverse)
4. Apply FileTransform adx_cdd_to_cid.spimtx (inverse)
5. Apply FileTransform adx_adx10_to_cdd.spimtx (inverse)

Note that this isn't possible in all cases (what if a lut or matrix is not invertible?), but conceptually it's a simple way to think about the design.

Summary

Configs and ColorSpaces are just a bookkeeping device used to get and ordered lists of Transforms corresponding to image color transformation.

Transforms are visible to the person AUTHORING the OCIO config, but are NOT visible to the client applications. Client apps need only concern themselves with Configs and Processors.

OCIO::Processors

A processor corresponds to a 'baked' color transformation. You specify two arguments when querying a processor: the colorspace_section you are coming from, and the colorspace_section you are going to.

Once you have the processor, you can apply the color transformation using the "apply" function. For the CPU veseion, first wrap your image in an ImageDesc class, and then call apply to process in place.

Example:

```
#include <OpenColorIO/OpenColorIO.h>
namespace OCIO = OCIO_NAMESPACE;

try
{
   // Get the global OpenColorIO config
   // This will auto-initialize (using $OCIO) on first use
   OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

   // Get the processor corresponding to this transform.
   // These strings, in this example, are specific to the above
   // example. ColorSpace names should NEVER be hard-coded into client
   // software, but should be dynamically queried at runtime from the library
   OCIO::ConstProcessorRcPtr processor = config->getProcessor("adx10", "aces");
   OCIO::ConstCPUProcessorRcPtr cpu = processor->getDefaultCPUProcessor();

   // Wrap the image in a light-weight ImageDescription
   OCIO::PackedImageDesc img(imageData, w, h, 4);

   // Apply the color transformation (in place)
   cpu->apply(img);
}
catch(OCIO::Exception & exception)
{
   std::cerr << "OpenColorIO Error: " << exception.what() << std::endl;
}
```

The GPU code path is similar. You get the processor from the config, and then query the shaderText and the lut3d. The client loads these to the GPU themselves, and then makes the appropriate calls to the newly defined function.

See *src/apps/ociodisplay* for an example.

Internal API

The Op Abstraction

It is a useful abstraction, both for code-reuse and optimization, to not relying on the transforms to do pixel processing themselves.

Consider that the FileTransform represents a wide-range of image processing operations (basically all of em), many of which are really complex. For example, the houdini lut format in a single file may contain a log convert, a 1d lut, and then a 3d lut; all of which need to be applied in a row! If we don't want the FileTransform to know how to process all possible pixel operations, it's much simpler to make light-weight processing operations, which the transforms can create to do the dirty work as needed.

All image processing operations (ops) are a class that present the same interface, and it's rather simple:

```
virtual void apply(float* rgbaBuffer, long numPixels)
```

Basically, given a packed float array with the specified number of pixels, process em.

Examples of ops include Lut1DOp, Lut3DOp, MtxOffsetOp, LogOp, etc.

Thus, the job of a transform becomes much simpler and they're only responsible for converting themselves to a list of ops. A simple FileTransform that only has a single 1D lut internally may just generate a single Lut1DOp, but a FileTransform that references a more complex format (such as the houdini lut case referenced above) may generate a few ops:

```
void FileFormatHDL::BuildFileOps(OpRcPtrVec & ops,
                         const Config& /*config*/,
                         const ConstContextRcPtr & /*context*/,
                         CachedFileRcPtr untypedCachedFile,
                         const FileTransform& fileTransform,
                         TransformDirection dir) const {

// Code omitted which loads the lut file into the file cache...

CreateLut1DOp(ops, cachedFile->lut1D,
                   fileTransform.getInterpolation(), dir);
CreateLut3DOp(ops, cachedFile->lut3D,
                   fileTransform.getInterpolation(), dir);
```

See (`src/core/*Ops.h`) for the available ops.

Note that while compositors often have complex, branching trees of image processing operations, we just have a linear list of ops, lending itself very well to optimization.

Before the ops are run, they are optimized. (Collapsed with appropriate neighbors, etc).

An Example

Let us consider the internal steps when getProcessor() is called to convert from ColorSpace 'adx10' to ColorSpace 'aces':

- The first step is to turn this ColorSpace conversion into an ordered list of transforms.

We do this by creating a single list of the conversions from 'adx10' to reference, and then adding the transforms required to go from reference to 'aces'. * The Transform list is then converted into a list of ops. It is during this stage luts, are loaded, etc.

CPU CODE PATH

The main list of ops is then optimized, and stored internally in the processor.

```
FinalizeOpVec(m_cpuOps);
```

During Processor::apply(...), a subunit of pixels in the image are formatted into a sequential rgba block. (Block size is optimized for computational (SSE) simplicity and performance, and is typically similar in size to an image scanline)

```
float * rgbaBuffer = 0;
long numPixels = 0;
while(true) {
   scanlineHelper.prepRGBAScanline(&rgbaBuffer, &numPixels);
   ...
```

Then for each op, op->apply is called in-place.

```
for(OpRcPtrVec::size_type i=0, size = m_cpuOps.size(); i<size; ++i)
{
   m_cpuOps[i]->apply(rgbaBuffer, numPixels);
}
```

After all ops have been applied, the results are copied back to the source

```
scanlineHelper.finishRGBAScanline();
```

GPU CODE PATH

1. The main list of ops is partitioned into 3 ordered lists:

- As many ops as possible from the BEGINNING of the op-list that can be done analytically in shader text. (called gpu-preops)
- As many ops as possible from the END of the op-list that can be done analytically in shader text. (called gpu-postops)
- The left-over ops in the middle that cannot support shader text, and thus will be baked into a 3dlut. (called gpu-lattice)

#. Between the first an the second lists (gpu-preops, and gpu-latticeops), we analyze the op-stream metadata and determine the appropriate allocation to use. (to minimize clamping, quantization, etc). This is accounted for here by interserting a forward allocation to the end of the pre-ops, and the inverse allocation to the start of the lattice ops.

See https://github.com/AcademySoftwareFoundation/OpenColorIO/blob/main/src/core/NoOps.cpp#L183

#. The 3 lists of ops are then optimized individually, and stored on the processor. The Lut3d is computed by applying the gpu-lattice ops, on the CPU, to a lut3d image.

The shader text is computed by calculating the shader for the gpu-preops, adding a sampling function of the 3d lut, and then calculating the shader for the gpu post ops.

5.2.4 Glossary

- Transform - a function that alters RGB(A) data (e.g transform an image from scene linear to sRGB)
- Reference space - a space that connects colorspaces
- Colorspace - a meaningful space that can be transferred to and from the reference space
- Display - a virtual or physical display device (e.g an sRGB display device)
- View - a meaningful view of the reference space on a Display (e.g a film emulation view on an sRGB display device)
- Role - abstract colorspace naming (e.g specify the "lnh" colorspace as the scene_linear role, or the color-picker UI uses color_picking role)
- Look - a color transform which applies a creative look (for example a per-shot neutral grade to remove color-casts from a sequence of film scans, or a DI look)

5.2.5 Presentations & Publications

- ASWF Open Source Days 2025 – Project Update video
- Color Interop Forum – "Color Space Encodings for Texture Assets and CG Rendering" Recommendation
- ASWF Open Source Days 2024 – NanoColor video
- ASWF Open Source Days 2024 – Project Update video
- OCIO honored with the SMPTE Workflow Systems Medal! article
- ASWF Open Source Days 2023 video
- OCIO receives the Pipeline Tool award at DigiPro 2022! video
- ASWF Open Source Days 2022 video
- HPA Tech Retreat 2022 "Color Processing with OCIO v2 and the Academy/ASC Common LUT Format" Slides & Tutorial
- OCIO v2 receives an Engineering Excellence award from the Hollywood Post Alliance! article
- SIGGRAPH 2021 course "Color management with OpenColorIO V2" video PDF ACM
- ASWF Open Source Days 2021 video
- DigiPro 2020 "The ASWF takes OpenColorIO to the Next Level" video PDF ACM
- ASWF Open Source Days 2020 video
- ASWF Open Source Days 2019 video
- Cinematic Color

5.3 Tutorials

We would like to have more tutorials, please help us add some! :)

An easy way to get involved is to join the #docs channel on *Slack*.

5.3.1 Baking LUT's

Sometimes it is necessary to write a color transform as a lookup-table file.

This is usually because an application does not natively support OCIO (unlike, say, Nuke which various OCIO nodes), but can load a LUT of some kind. This would currently include applications like Autodesk Flame, Adobe Photoshop, SideFX's MPlay (Houdini's "Image Viewer").

Remember that baking a LUT is not a perfect solution. Different LUT formats have various limitations. Certain applications might apply LUT's differently (often incorrectly), and some formats simply cannot accurately represent certain transforms. Others might require carefully selecting shaper spaces and so on.

Be sure to carefully test the generated LUT in the target application. Burning the LUT into a test image (such as Marcie!), and carefully comparing to a reference is often the only way to be sure a LUT is correct.

Config-based baking

This section assumes you have a working OCIO config.

The config can either be specified by setting the OCIO environment variable:

```
bash$ export OCIO=~/path/to/spi-vfx/config.ocio
```

Alternatively the config can be specified as a command-line argument to the ociobakelut command, `--iconfig ~/path/to/spi-vfx/config.ocio`

These examples will use the *spi-vfx* config, specifically the following colorspaces

- `lnf` - scene-referred linear light colorspace (reference colorspace)
- `lg10` - film log colorspace (0-1 colorspace)
- `srgb8` - sRGB display colorspace

Remember these are just for the examples - you can of course use any config and any colorspaces

Log-input display LUT

Say we have a `lg10` image in MPlay (maybe a ".cin" film scan), and wish to view it in our main display colorspace, `srgb8`

The available formats are listed in the `ociobakelut --help` - for MPlay, we use the "houdini" format (see *the FAQ* for a more detailed list)

So, to create a LUT that transforms from `lg10` to `srgb8`:

```
bash$ ociobakelut --format houdini --inputspace lg10 --outputspace srgb8 log_to_display.
↪lut
```

We could then load this LUT into MPlay and view a lg10 image correctly! (note that by MPlay tries to linearize ".cin" files by default, which can be disabled in the preferences, "Convert form 10bit Log")

For most other applications, we could simply change the `--format`

Shaper spaces

Before we create a LUT to view linear-light images, it's important to understand shaper-spaces and preluts.

The two main types of LUT's are 1D and 3D. Such LUT formats typically require input values in the 0.0-1.0 range. Such a LUT would be unsuitable for linear-light values input images (where values are often over 1)

To address this, various LUT formats contain a smaller "prelut" (or "shaper LUT") which is applied before the main LUT. This is used to transform the input values into a 0-1 range (typically a linear-to-log type transform)

In terms of color-transforms, the prelut transforms from "input-space" to "shaper-space", then the main LUT transforms from "shaper-space" to "output-space"

Some formats do not support such a shaper LUT - these are typically used in applications which do not work with floating-point images (e.g Lustre often works with 10-bit integer DPX's, so it's natively supported "–format lustre" (3DL) format has no need for a prelut)

Linear-light input display LUT

With shaper-spaces explained, lets say we have a `lnf` linear light image in MPlay, and wish to view it in the `srgb8` colorspace.

To create this LUT accurately, without clipping, we will use the LUT's prelut to transform from `lnf` to `lg10`, then the 3D LUT will transform from `lg10` to `srgb8`

Sounds complicated, but the command is almost the same as before, just with the `--shaperspace` argument (and `--inputspace` changed, of course):

```
bash$ ociobakelut --format houdini --inputspace lnf --shaperspace lg10 --outputspace␣
↪srgb8 lin_to_display.lut
```

Allocation-based prelut

If your *allocation variables* are setup correctly, you can omit the `--shaperspace` argument, and a prelut will be automatically created based on the allocation vars (see the linked page for more information)

Since the colorspaces in the config we are using (*spi-vfx*) have their allocation variables set correctly, we could simplify the `lnf` to `srgb8` bake command:

```
bash$ ociobakelut --format houdini --inputspace lnf --outputspace srgb8 lin_to_display_
↪allocbased.lut
```

This command creates a very different prelut to the explicitly specified `--shaperspace lg10` in the previous example. Explicitly specifying a shaper can produce better results, however the allocation-based prelut usually works nicely, and is convenient

Note that allocation-var based preluts is baker-format dependent, and not all formats currently implement them

Config-free baking

You can perform baking without using a OCIO config. This means you don't have to create a temporary config just to, say, convert from one LUT format to another.

Converting between formats

Say we have a houdini LUT named `log_to_display.lut`. To convert this to a Flame compatible 3dl file, simply run:

```
ociobakelut --format flame --lut log_to_display.lut for_flame.3dl
```

Reversing a LUT

You can apply a LUT in reverse, and write this to a new LUT:

```
bash$ ociobakelut --format flame --invlut logtosrgb.3dl srgbtolog.3dl
```

Creating a grade LUT

You can create a LUT which applies CDL-compliant grades:

```
ociobakelut --format cinespace --slope 1.2 1.0 0.9 mygrade.csp
```

Combining options

These options can be used together, or used multiple times.

For example, to perform a slope offset, then apply "mylut.csp", saving it out for Lustre:

```
bash$ ociobakelut --format lustre --slope 2.0 1.5 0.4 --lut mylut.csp output.3dl
```

ICC profiles (Photoshop)

Photoshop is very focused around print and graphic-design, industries with very different color management concerns to modern feature-film VFX. As such, it does not fit smoothly into a VFX workflow.

The main issue is current versions of Photoshop (CS5) are only practical for working with 16-bit integer images (not floating point/linear-light images as is common in compositing software)

The second issue is there is no simple way to load a simple 1D or 3D LUT into Photoshop (and its API does not make this easy either!)

A working space

First, we need to decide on a colorspace to use for the images in Photoshop. This is the colorspace in which matte-paintings will be performed (likely a different colorspace that used for texture-painting, as these have different requirements)

The working space should be a "0-1 colorspace", reversible, and for matte-paintings ideally allow painting values over "diffuse white" (in other words, to paint values over 1.0 when converted to linear-light in comp)

This is a facility-dependent workflow choice.

For this example we will use the `vd16` colorspace, as described by the *spi-vfx*

Creating display LUT

"Proofing profiles" in Photoshop can be used in a very similar way to a display LUT in applications like Nuke. This proof-profile can be used to apply a 3D color transform from the working-space to a display space (e.g transform from `vd16` to `srgb8` in the spi-vfx config)

These proofing-profiles are ICC profiles - a rather print-specific technology and relatively complex format

Luckily, `ociobakelut` can be used to create these… but, first, there are some important considerations:

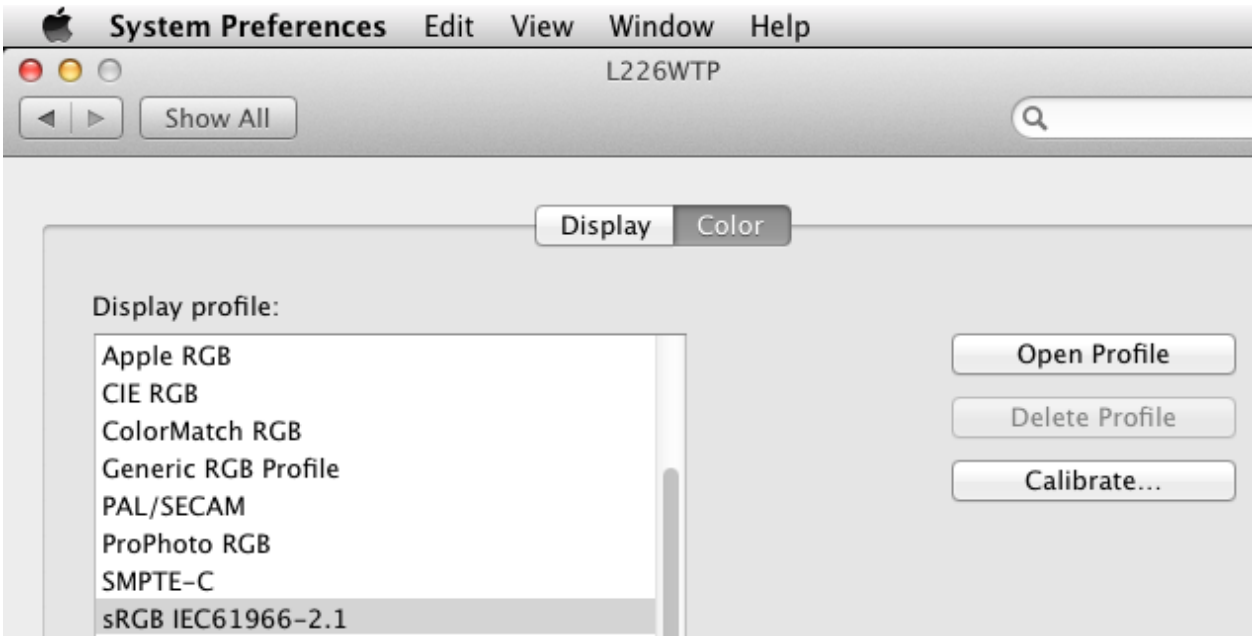
It is important to match the `--displayicc` option to the profile used for the display.

Secondly, Photoshop has a lot of print-focused color-management options, some of which can cause problems.

Determine display ICC

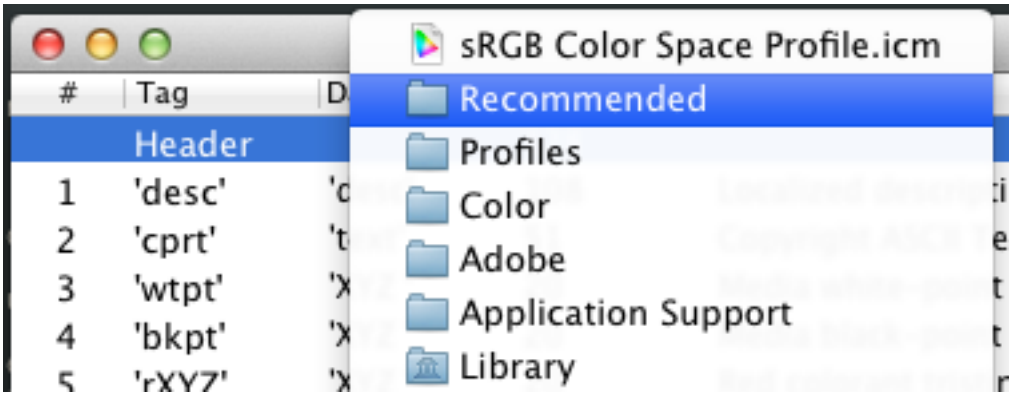
On OS X, launch "System Preferences", open "Displays" and click "Color" tab. The currently active ICC profile is selected.

If you just want something simple that "probably matches" a Linux machine, say, it is easiest to uncheck "Show profiles for this display only" and select the "sRGB IEC61966-2.1" profile. You can skip the rest of this section in this case.



However, if you have a specific display-profile selected (maybe created by monitor-calibration software), you should do the following:

Click "Open Profile", and right-click the icon in the top of the window, and click the folder:



This reveals the current profile in Finder. You can drag the file onto a Terminal.app window to get the full path (or, type it manually)

Create the ICC profile

Almost done now. We can write the ICC profile!

The full command is, using our example colorspaces of `vd16` and `srgb8`:

```
bash$ ociobakelut --format icc --inputspace vd16 --outputspace srgb8 --displayicc /path/
↪to/my/monitorprofile.icc --description "vd16 to srgb8"
```

The first three options are the same as any other LUT:

```
bash$ ociobakelut --format icc --inputspace vd16 --outputspace srgb8 [...]
```

Then we specify the display ICC profile:

```
[...] --displayicc /path/to/my/monitorprofile.icc [...]
```

We can set the description (shown in Photoshop), and as the last argument, specify:

```
[...] --description "vd16 to srgb8"  [...]
```

Finally an argument for the output file:

[...] vd16_to_srgb.icc

If you selected the "sRGB IEC61966-2.1" display profile, you can omit the `--displayicc` argument (it defaults to an standard sRGB profile):

```
bash$ ociobakelut --format icc --inputspace vd16 --outputspace srgb8 --description "vd16␣
↪to srgb8"
```

Loading the "display LUT"

Last step is to load the ICC profile into Photoshop, and enable it.

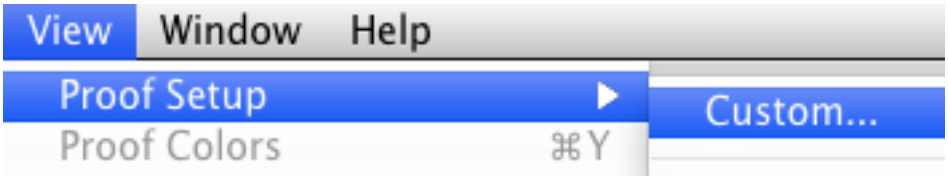
On OS X, these can be put into:

```
/Library/ColorSync/Profiles/
```

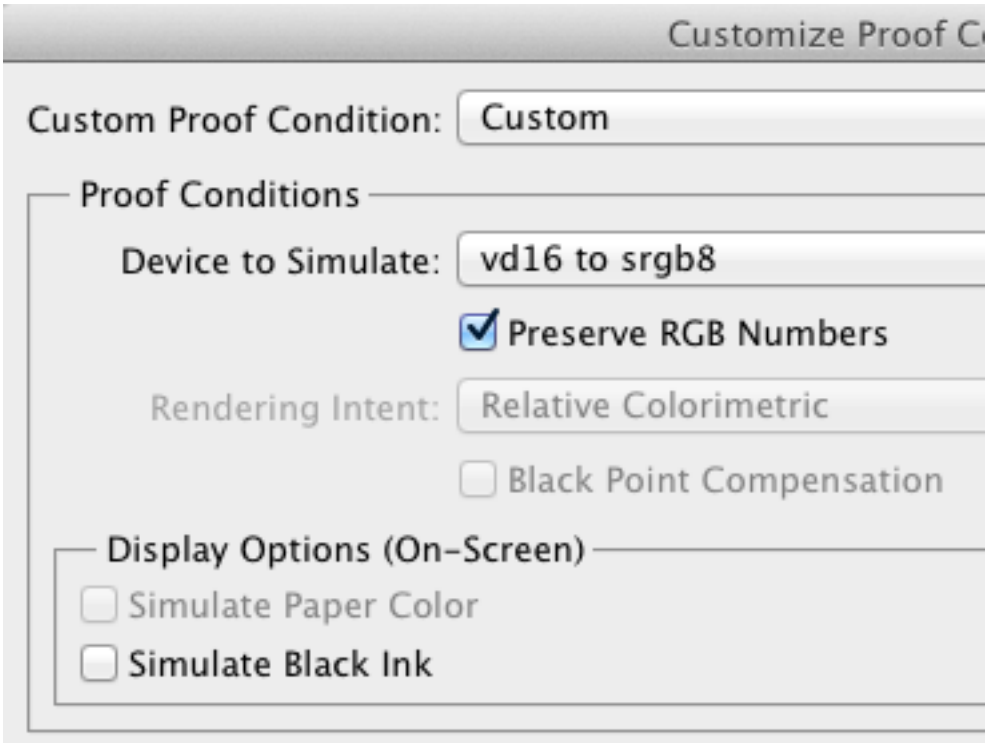
(or the equivalent directory in your home-directory)

On Windows, right-click the profile and select "Install profile"

Then on either platform, click "View > Proof Setup > Custom…"



Select your profile from the "Device to simulate" dropdown (the name is what you supplied with `--description`):



As pictured, selecting "Preserve RGB numbers", and deselecting "Simulate Black Ink" is a good starting point (see the next section on "Other color settings")

Finally, you can load an image in your working space, and click "View > Proof Colors", or hit cmd+y (or ctrl+y)

When active, the profile name is shown in the window title (e.g "…. (RGB/16#/vd16 to srgb8", where the part after the "#/" is the profile name, "RGB/16" indicates the current image mode)

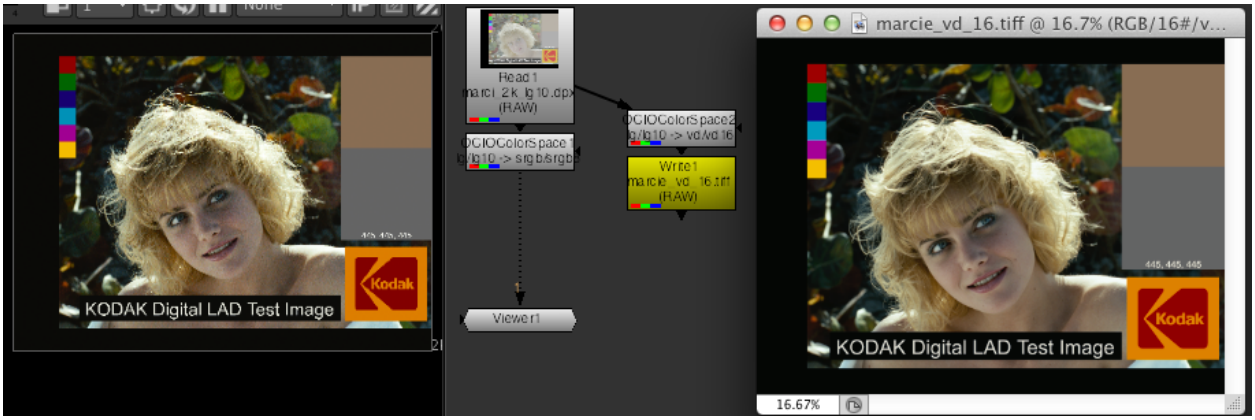
Other color settings

(note this guide is based on Photoshop CS5, and written while checking the OS X version, although most of these apply similarly on Windows 7)

It is usually possible to get a matte-painting to look identical in Photoshop as it does in a more VFX-friendly application such as Nuke.

However, as mentioned Photoshop has a lot of color-management related options, many of which can impair the match between it and other applications. The operating system also has some controls (as seen before with the ColorSync display profile)

The settings that require tweaking have a tendency to change with OS versions, Photoshop versions and the phase of the moon. The only way to be sure is to compare Photoshop side-by-side with a LUT-reference-image (ideally toggling between Photoshop and Nuke):



The most important settings are in the "View > Proof Setup > Custom . . . " menu.

The recommended "Preserve RGB" setting works sometimes. Other times disabling "Preserve RGB Numbers" and selecting "Rendering Intent: Relative Colorimetric" can be closer.

It is safest to not assign a profile to the images you are working on - this is done by clicking "Edit > Assign Profile", and selecting "Don't Color Manage This Document".

In closing, of course none of this matters if you don't have a calibrated monitor!

5.3.2 Contributing

Coming soon . . .

5.4 Guides

5.4.1 Using OCIO

5.4.2 Environment Variables

The following environment variables may be used with OpenColorIO:

OCIO

Specifies the file path of the config file to be used. (This is necessary for some of the command-line tools but most applications provide a way of setting this from the user interface.)

OCIO_ACTIVE_DISPLAYS

Overrides the active-displays list from the config file and reorders them. Colon-separated list of displays, e.g `sRGB:P3`

OCIO_ACTIVE_VIEWS

Overrides the active-views list from the config file and reorders them. Colon-separated list of view names, e.g `internal:client:DI`

OCIO_INACTIVE_COLORSPACES

Overrides the inactive_colorspaces list from the config file. Colon-separated list of color spaces, e.g `previousColorSpace:tempSpace`

OCIO_LOGGING_LEVEL

Configures OCIO's internal logging level. Valid values are `none`, `warning`, `info`, or `debug` (or their respective numeric values `0`, `1`, `2`, or `3` can be used)

Logging output is sent to STDERR output by default.

OCIO_OPTIMIZATION_FLAGS

Overrides the optimization settings being used by an application, for troubleshooting purposes. The complete list of flags is in OpenColorTypes.h.

OCIO_USER_CATEGORIES

Specify the color space categories that the application should show in color space menus. Note that this requires the application developer to implement support for categories (the easiest way is to use the code in apphelpers/ColorSpaceHelpers.h).

5.4.3 Tool overview

The following command-line tools are provided with OpenColorIO.

The –help argument may be provided for info about the arguments and most tools use the -v argument for more verbose output.

Many of the tools require you to first set the OCIO environment variable to point to the config file you want to use.

Note that some tools depend on OpenEXR or OpenImageIO and other libraries:

- ociolutimage: (OpenEXR or OpenImageIO)
- ociodisplay: (OpenEXR or OpenImageIO), OpenGL, GLEW, GLUT
- ocioconvert: (OpenEXR or OpenImageIO), OpenGL, GLEW, GLUT

ocioarchive

This command-line tool allows you to convert a config and its external LUT files into an OCIOZ archive file. A .ocioz file may be supplied to any command that takes the path to a config or set as the OCIO environment variable.

This example creates a file called myarchive.ocioz:

```
$ ocioarchive myarchive --iconfig myconfig/config.ocio
```

This command will expand it back out:

```
$ ocioarchive --extract myarchive.ocioz
```

The –list option may be used to see the contents of a .ocioz file.

Note: Archive files generated on Windows machines in OCIO 2.4 or earlier should be regenerated in OCIO 2.5 or higher due to a bug in the third-party library being used, otherwise they may not open properly on Linux or macOS systems.

ociocheck

This is a command-line tool which shows an overview of an OCIO config file, and checks for obvious errors.

For example, the following shows the output of a config with a typo - the colorspace used for `compositing_log` is incorrect:

```
$ ociocheck --iconfig example.ocio

OpenColorIO Library Version: 0.8.3
OpenColorIO Library VersionHex: 525056
Loading example.ocio

** General **
Search Path: luts
Working Dir: /tmp

Default Display: sRGB
Default View: Film

** Roles **
ncf (default)
lnf (scene_linear)
NOT DEFINED (compositing_log)

** ColorSpaces **
lnf
lgf
ncf
srgb8 -- output only

ERROR: Config failed validation. The role 'compositing_log' refers to a colorspace, 'lgff
↪', which is not defined.

Tests complete.
```

It cannot verify the defined color transforms are "correct", only that the config file can be loaded by OCIO without error. Some of the problems it will detect are:

- Duplicate colorspace names
- References to undefined colorspaces
- Required roles being undefined
- At least one display device is defined
- No v2 features are used in a v1 config
- Validates the structure and usage of interop IDs

As with all the OCIO command-line tools, you can use the *–help* argument to read a description and see the other arguments accepted:

```
$ ociocheck --help
ociocheck -- validate an OpenColorIO configuration

usage:  ociocheck [options]

    --help        Print help message
    --iconfig %s  Input .ocio configuration file (default: $OCIO)
    --oconfig %s  Output .ocio file
```

ociochecklut

The ociochecklut tool will load any LUT type supported by OCIO and print any errors or warnings encountered. An RGB or RGBA value may be provided and will be evaluated through the LUT and the result printed to the console. Alternatively, the -t option will send a predefined set of test RGB values through the LUT:

```
$  ociochecklut -v acescct_to_rec709.clf  0.18 0.18 0.18

OCIO Version: 2.0.0dev

Input  [R G B]: [      0.18      0.18       0.18]
Output [R G B]: [0.05868321 0.0586832 0.05868318]
```

The –gpu argument may be used to evaluate using the GPU renderer rather than the CPU renderer.

ociobakelut

A command-line tool which bakes a color transform into various color lookup file formats ("a LUT")

This is intended for applications that have not directly integrated OCIO, but can load LUT files

If we want to create a `lnf` to `srgb8` viewer LUT for Houdini's MPlay:

```
$ ociobakelut --inputspace scene_linear --shaperspace lg10 --outputspace srgb8 --format␣
↪houdini houdini__lnf_to_lg10_to_srgb8.lut
```

The `--inputspace` and `-outputspace` options specify the colorspace of the input image, and the displayed image.

Since a 3D LUT can only practically operate on 0-1 (e.g a Log image), the `--shaperspace` option is specified. This uses the Houdini LUT's 1D "pretransform" LUT to do "lnf" to "lg10", then the 3D LUT part to go from "lg10" to "srgb8" (basically creating a single file containing a 1D linear-to-log LUT, and a 3D log-to-sRGB LUT)

To make a legacy log to sRGB LUT in 3dl format, the usage is similar, except the shaperspace option is omitted, as the input colorspace does not have values outside 0.0-1.0 (being a Log space):

```
$ ociobakelut --inputspace lg10 --outputspace srgb8 --format flame flame__lg10_to_srgb.
↪3dl
```

See the *Supported LUT Formats* section for a list of formats that support baking, and see the *Baking LUT's* for more information on baking LUTs.

The ociobakelut command supports many arguments, use the –help argument for a summary.

ocioconvert

Loads an image, applies a color transform, and saves it to a new file.

The ocioconvert tool applies either an aribtrary LUT, or a complex OpenColorIO transform. OCIO transforms can be from an input color space to either an output color space or a (display,view) pair.

Both CPU (default) and GPU renderers are supported. The –gpuinfo argument may be used to output the shader program used by the GPU renderer.

Uses OpenImageIO or OpenEXR for opening and saving files and modifying metadata. Supported formats will vary depending on the use of OpenImageIO. The interop ID, if available, is written to the header of OpenEXR files.

Use the –help argument for more information on to the available options.

ociocpuinfo

This tool simply prints information about the CPU in your computer. OCIO will switch in different versions of the apply functions based on which SIMD instruction sets are supported by a given CPU, so this is provided as a potential trouble-shooting tool.

ociodisplay

An example image viewer demonstrating the OCIO C++ API.

Uses OpenImageIO or OpenEXR to load images, and displays them using OCIO and typical viewer controls (scene-linear exposure control and a post-display gamma control).

May be useful to users to quickly check a color space configuration.

NOTE: This program is not a very good example of how to build a UI. For example, it assumes each display has the same views, which is often not the case. Also, it does not leverage any of the new OCIO v2 features.

pyociodisplay

The pyociodisplay tool is a minimal image viewer implementation demonstrating use of the OCIO GPU renderer in a Python application. It requires downloading a few dependencies before use. For more information, please see the README.

ociolutimage

The ociolutimage tool converts a 3D LUT to or from an image.

Image containers are occasionally used for encoding and exchanging simple color lookup data where standard LUT formats are less feasible. The ociolutimage tool offers an arguably "artist-friendly", WYSIWYG workflow for creating LUTs representing arbitrary color transforms.

The workflow is a three step process:

```
1. Generate an identity lattice image with ociolutimage --generate
2. Apply color transforms to the generated image (e.g., in a DCC application)
3. Extract LUT data from the modified RGB lattice values with
   ociolutimage --extract
```

See also:

Nuke's CMSTestPattern and GenerateLUT nodes are analogous to the ociolutimage –generate and –extract options, respectively. Applications such as Lattice provide similar functionality.

ociomakeclf

The ociomakeclf tool converts a LUT into Academy/ASC Common LUT Format (CLF):

```
$ ociomakeclf lut_file.cube lut_file.clf
```

The –csc argument may be used to convert the LUT into an ACES compliant Look Modification Transform (LMT) that may be referenced from an ACES Metadata File. An ACES LMT requires ACES2065-1 color space values on input and output. The –csc argument is used to specify the standard color space that the Look LUT expects on input and output and the tool will prepend a transform from ACES2065-1 to the LUT color space and postpend a transform from that color space back to ACES2065-1:

```
$ ociomakeclf my_ACEScct_look.cube my_LMT.clf --csc ACEScct
```

The –list argument will print out all of the standard ACES color spaces that are supported as –csc arguments.

ociomergeconfigs

The ociomergeconfigs tool processes an OCIOM file to merge one or more config files:

```
$ ociomergeconfigs mergeFile.ociom --out mergedConfig.ocio
```

The OCIOM file identifies the source configs and controls the merge process. Please refer to the API documentation for the ConfigMergingParameters class for an explanation of the OCIOM file format, the available merge strategies and the parameters available to control the merge process.

Additional documentation on this feature will be forthcoming.

ocioperf

The ocioperf tool allows you to benchmark the performance of a given color transformation on your hardware. Please use the –help argument for a description of the options.

The metric used for assessing performance is the time taken to apply a transform to an image with respect to each pixel, to each line, or to the entire image plane (or all three). By default, each test is run ten times.

Transforms are either provided as an external file or specified in the active config (i.e., the config pointed to by the OCIO environment variable).

Examples:

```
$ ocioperf -displayview ACEScg sRGB 'Show LUT' -iter 20 -image test.exr
# Measures an ACEScg -> sRGB / 'Show LUT' DisplayViewTransform applied to each
# pixel of 'test.exr' twenty times.

$ ocioperf -transform my_transform.ctf -out f32 -image meow.jpg
# Measures 'my_transform.ctf' applied to the whole 'meow.jpg' image and output
# as 32-bit float data ten times.

$ ocioperf -colorspaces 'LogC AWG' ACEScg -test 1 -image marcie.dpx
# Measures a 'LogC AWG' -> ACEScg ColorSpaceTransform applied to each line of
# 'marcie.dpx' ten times.
```

ocioview

This is a new GUI tool for inspecting and editing config files. It is currently an alpha release and we are looking for contributors to extend it or provide tutorials. Please see the README in apps/ocioview for details about installation.

ociowrite

The ociowrite tool allows you to serialize a color transformation to an XML file. This is useful for troubleshooting and also to be able to send a complete OCIO color conversion as a single file.

Note that this command does not do any baking of the transform into another format and so should give identical results to the original.

The –colorspaces argument specifies the source and destination color spaces for a ColorSpaceTransform and the –file argument specifies that name of the output file. The OCIO environment variable is used to specify the config file to be used.

The two file formats supported are CTF and CLF and this is selected by the extension you provide to the –file argument. The CTF format is recommended because it is able to represent all OCIO transforms and operators. The CLF format is also allowed since it has wider support in non-OCIO applications but the tool will not write the file if the transformation would require an operator that is not supported by CLF.

Here is an example:

```
$ export OCIO=/path/to/the/config.ocio
$ ociowrite --colorspaces acescct aces2065-1 --file mytransform.ctf
```

pyocioamf

The pyocioamf tool is an initial attempt to support the ACES Metadata File (AMF) format. This Python script will take an AMF file and produce an OCIO CTF file that implements its color pipeline. The CTF file may be applied to images using tools such as *ociconvert*. For more information, please see the README.

5.4.4 Supported LUT Formats

Ext	Details	Notes
3dl	Discreet legacy 3D-LUT	Read + Write Support. Supports shaper + 3D-LUT
ccc	ASC CDL ColorCorrectionCollection	Full read support.
cc	ASC CDL ColorCorrection	Full read support.
cdl	ASC CDL ColorCorrection	Full read support.
clf	Academy/ASC Common LUT Format	Full read + write support.
ctf	Autodesk Color Transform Format	Full read + write support. Provides lossless serialization of arbitrary OCIO transforms.
csp	Cinespace (Rising Sun Research) LUT. Spline-based shaper LUT, with either 1D or 3D-LUT.	Read + Write Support. Shaper is resampled into simple 1D-LUT with 2^16 samples.
cub	Truelight format. Shaper Lut + 3D	Full read support.
cube	Iridas format. Either 1D or 3D Lut.	Read support.
cube	Resolve format.	Read support.
lut	Discreet legacy 1D-LUT	Read support.
hdl	Houdini. 1D-LUT, 3D-LUT, 1D shaper Lut	Only 'C' type is supported. Need to add R G B A RGB RGBA ALL. No support for Sampling tag. Header fields must be in strict order.
icc/ic	ICC profile format	Read support for basic monitor profiles.
look	IRIDAS .look	Read baked 3D-LUT embedded in file. No mask support.
mga/r	Pandora 3D-LUT	Full read support.
spi1d	1D-LUT format. Imageworks native format. HDR friendly, supports arbitrary input and output domains	Full read support.
spi3d	3D-LUT format. Imageworks native format.	Full read support.
spimt	3x3 matrix + color offset. Imageworks native matrix format.	Full read support.
vf	Inventor 3D-LUT.	Read support for 3D-LUT data and global_transform element

Note: Some LUT formats include a "shaper" before a 3D-LUT intended to help the 3D-LUT work better with input spaces that are not perceptually uniform. These shapers are sometimes not actually LUTs and OCIO needs to convert them into one of the existing operators such as a LUT1D. In some cases this is done with linear interpolation, which does not work well with coarsely sampled shapers. In other cases (such as CSP format) spline interpolation is used, but the resulting LUT1D may not have fine enough sampling in the shadows to adequately represent the original shaper. If you are able to use them, the CLF and CTF formats are recommended since they allow very powerful shaper options that may be converted exactly into OCIO operators.

5.4.5 Compatible Software

The following software all supports OpenColorIO (to varying degrees).

If you are interested in getting OCIO support for an application not listed here, please contact the ocio-dev mailing list.

If you are a developer and would like assistance integrating OCIO into your application, please contact ocio-dev as well.

3ds Max (Autodesk)

Website : https://www.autodesk.com/products/3ds-max/overview

Supported Version: >= 2024

Documentation :

- Color Management
- OCIO color management in Autodesk 3DS Max 2024.1

After Effects

An OpenColorIO plugin is available for use in After Effects.

Website: http://www.adobe.com/products/aftereffects.html

Documentation :

- See src/aftereffects if you are interested in building your own OCIO plugins.
- Pre-built binaries are also available for download, courtesy of fnordware.

Anchorpoint

Anchorpoint is a file browser and asset manager for VFX, animation and real-time projects. It supports OCIO configs to display OpenEXR with the correct color profile.

Website: https://www.anchorpoint.app

Documentation :

- Display OpenEXR with OpenColorIO

Arnold (Autodesk)

Website : https://www.autodesk.com/products/arnold/overview

Documentation :

- Color Management

Blender

Blender has integrated OCIO as part of it's redesigned color management system.

Website: http://www.blender.org/

Supported Version: >= 2.64

Documentation:

- Blender 2.64 Release Notes
- Color Management

C++

The core OpenColorIO API is available for use in C++. Minimal code examples are also available in the source code distribution.

Website: http://github.com/AcademySoftwareFoundation/OpenColorIO/tree/main/include/OpenColorIO

Examples:

- ocioconvert
- ociodisplay

Documentation:

- *Developing with OCIO*
- developers-usageexamples

cineSync (ftrack)

Website : https://cospective.com/cinesync/

Documentation :

- Colour Grading

Clarisse (Isotropix)

Website: https://www.isotropix.com/products

Documentation:

- Managing Looks with OpenColorIO

ColorBox (AJA)

In-line HDR/SDR Algorithmic and LUT Color Transforms. Available as a standalone unit (ColorBox) and also as an openGear card (OG-ColorBox).

Website : https://www.aja.com/products/colorbox

Website : https://www.aja.com/products/og-colorbox

Supported Version : >= 3.0

CryEngine (CryTek)

CryEngine is a real-time game engine, targeting applications in the motion-picture market. While we don't know many details about the CryEngine OpenColorIO integration, we're looking forward to learning more as information becomes available.

Website: https://www.cryengine.com/

Disguise

Disguise is an integrated hardware and software platform powering the world's biggest live events, immersive experiences and virtual production stages. OpenColorIO is integrated into the Designer software, allowing users to manage the color spaces of all content sources and physical output devices.

Website: http://disguise.one/

Supported version: >= 29.1

Documentation:

- OpenColorIO Support
- Color Management

DJV

Website: https://darbyjohnston.github.io/DJV/

Flame (Autodesk)

Website: Autodesk Flame

Supported version: >= 2026

Documentation:

- OCIO in Flame

Gaffer

Gaffer is a node based application for use in CG and VFX production, with a particular focus on lighting and look development.

Website: http://www.gafferhq.org/

Guerilla Render

Composed of Guerilla Station and Guerilla Render, Guerilla is a Production-Proven Look Development, Assembly, Lighting and Rendering Solution designed for the Animation and VFX industries.

Website: http://guerillarender.com/

Supported version: >= 2.2

Documentation:

- OpenColorIO Support

- OCIO Management

Hiero (Foundry)

Hiero ships with native support for OCIO in the display and the equivalent of Nuke's OCIOColorSpace in the Read nodes. It comes with the "nuke-default" OCIO config by default, so the Hiero viewer matches when sending files to Nuke for rendering.

Website: https://www.foundry.com/products/hiero

Supported version: >= 1.0

Houdini (SideFX)

Website: https://www.sidefx.com

Supported Version: >= 16

Documentation:

- OpenColorIO Support
- What's new in Houdini 16

Katana (Foundry)

Website: http://www.thefoundry.co.uk/products/katana

Documentation:

- Managing Color
- Color Nodes

Krita

Krita now support OpenColorIO for image viewing, allowing for the accurate painting of float32/OpenEXR imagery.

Website: http://www.krita.org/

Documentation :

- Krita Starts Supporting OpenColorIO.

Mari (Foundry)

Website: https://www.foundry.com/products/mari

Supported Version: >= 1.4v1

Documentation:

- Managing Colors in Mari
- Color Management

Maya (Autodesk)

Autodesk Maya is a 3D computer animation, modeling, simulation, and rendering software.

Website: https://www.autodesk.com/products/maya/overview

Supported Version: >= 2016

Documentation:

- Get Started with Color Management
- Color Management in Maya: Setting up a scene
- Color Management in Maya: Input Spaces
- Color Management in Maya: Previewing and Rendering
- Color Management in Maya: Color Picking
- Color Management in Maya: ACES default

Mocha Pro (Boris FX)

Website: https://borisfx.com/products/mocha-pro

Supported Version: >= 2020

Documentation:

- Color Management

Modo (Foundry)

Website: https://www.foundry.com/products/modo

Documentation:

- Color Management

mrViewer

mrViewer is a professional flipbook player, hdri viewer and video/audio playback tool. It supports OCIO input color spaces in images as well as display/view color spaces.

Website: https://mrviewer.sourceforge.io

Documentation:

- mrViewer Features

Natron

Natron is an open source 2D compositor that ships with native support for OCIO. Standard configs are included however users can also point to custom configs in the color management section of the user preferences.

Website : https://natrongithub.github.io/

Documentation :

- Color Nodes

Nuke (Foundry)

Nuke ships with native support for OpenColorIO. There is also an available "nuke-default" OCIO config, which matches the built-in Nuke color processing. This profile is useful for those who want to mirror the native Nuke color processing in other applications (the underlying equations are also provided as python code in the config as well).

Website : https://www.foundry.com/products/nuke

Supported Version: >= 6.3v7

Documentation:

- OCIO Color Management
- Color Nodes

OpenImageIO

OIIO's C++ and Python bindings include several methods for applying color transforms to whole images, notably functions in the ImageBufAlgo namespace including **colorconvert()**, **ociolook()**, **ociodisplay()**, **ociofiletransform()**. These are also available as part of the *oiiotool* command line utility (–colorconvert, –ociolook, –ociodisplay, –ociofiletransform) and the *maketx* utility for preparing textures also supports **–colorconvert**. From C++, there is additional low-level functionality in the header **OpenImageIO/color.h** that are wrappers for accessing underlying OCIO color configurations and doing color processing on individual pixel values.

Website : http://openimageio.org

PhotoFlow

PhotoFlow supports OCIO via a dedicated tool that can load a given configuration and apply the available color transforms. So far the tool has been tested it with the Filmic and ACES configs.

Website : https://github.com/aferrero2707/PhotoFlow

Photoshop

OCIO can be enabled via OpenColorIO settings. For more details see OpenColorIO and tools with 32-bit mode.

Photoshop Fnordware plugin

An OpenColorIO plugin is also available for use in Photoshop. The plug-in can perform color operations to an image as a filter and can also export LUTs and ICC profiles to be used by Photoshop.

Plugin binaries are available for download, courtesy of fnordware.

Python

The OpenColorIO API is available for use in Python.

Website: https://pypi.org/project/opencolorio/

Documentation:

- *Developing with OCIO*
- developers-usageexamples

RV (Autodesk)

Website : https://www.autodesk.com/products/flow-production-tracking/overview

OpenRV : https://github.com/AcademySoftwareFoundation/OpenRV

Supported Version: >= 4

Documentation :

- For more details, see the OpenColorIO section of the RV User Manual.

Silhouette (Boris FX)

OCIO is natively integrated in Silhouette. Full support is provided for both image import/export, as well as image display.

Website : https://borisfx.com/products/silhouette/

Supported Version: >= 4.5

Substance Designer (Adobe)

Website: https://www.substance3d.com/products/substance-designer/

Supported version: >= 2019.3

Documentation:

- Color Management with OpenColorIO
- Color Management

TouchDesigner (Derivative)

Website: https://derivative.ca/

Documentation:

- OpenColorIO TOP

Unreal Engine (Epic Games)

Website : https://unrealengine.com

Supported Version : >= 4.22

Documentation :

- OCIO Plugin API
- Unreal Engine 4.22 Release Notes

Vegas Pro (Magix)

Vegas Pro uses OpenColorIO, supporting workflows such as S-log footage via the ACES colorspace.

Website : http://www.sonycreativesoftware.com/vegaspro

Supported Version: >= 12

V-Ray (Chaos Group)

Website : https://chaosgroup.com

Documentation :

- OpenColorIO support
- VRayTexOCIO

VRED (Autodesk)

Website : https://www.autodesk.com/products/vred/overview

Supported Version: >= 2025.3

Documentation :

- Autodesk VRED 2025.3

Apps w/icc or luts

flame (.3dl), lustre (.3dl), cinespace (.csp), houdini (.lut), iridas_itx (.itx) photoshop (.icc)

Export capabilities through ociobakelut:

```
$ ociobakelut -- create a new LUT or icc profile from an OCIO config or lut file(s)
$
$ usage:  ociobakelut [options] <OUTPUTFILE.LUT>
$
$ example:  ociobakelut --inputspace lg10 --outputspace srgb8 --format flame lg_to_srgb.
↪3dl
$ example:  ociobakelut --lut filmlut.3dl --lut calibration.3dl --format flame display.
↪3dl
$ example:  ociobakelut --lut look.3dl --offset 0.01 -0.02 0.03 --lut display.3dl --
↪format flame display_with_look.3dl
$ example:  ociobakelut --inputspace lg10 --outputspace srgb8 --format icc ~/Library/
↪ColorSync/Profiles/test.icc
$ example:  ociobakelut --lut filmlut.3dl --lut calibration.3dl --format icc ~/Library/
↪ColorSync/Profiles/test.icc
$
$
$ Using Existing OCIO Configurations
$     --inputspace %s      Input OCIO ColorSpace (or Role)
$     --outputspace %s     Output OCIO ColorSpace (or Role)
$     --shaperspace %s     the OCIO ColorSpace or Role, for the shaper
$     --iconfig %s         Input .ocio configuration file (default: $OCIO)
$
$ Config-Free LUT Baking
$     (all options can be specified multiple times, each is applied in order)
$     --lut %s             Specify a LUT (forward direction)
$     --invlut %s          Specify a LUT (inverse direction)
$     --slope %f %f %f     slope
$     --offset %f %f %f    offset (float)
$     --offset10 %f %f %f  offset (10-bit)
$     --power %f %f %f     power
$     --sat %f             saturation (ASC-CDL luma coefficients)
$
$ Baking Options
$     --format %s          the lut format to bake: flame (.3dl), lustre (.3dl),
$                          cinespace (.csp), houdini (.lut), iridas_itx (.itx), icc (.
↪icc)
$     --shapersize %d      size of the shaper (default: format specific)
$     --cubesize %d        size of the cube (default: format specific)
$     --stdout             Write to stdout (rather than file)
$     --v                  Verbose
$     --help               Print help message
$
$ ICC Options
$     --whitepoint %d      whitepoint for the profile (default: 6505)
$     --displayicc %s      an icc profile which matches the OCIO profiles target display
$     --description %s     a meaningful description, this will show up in UI like␣
↪photoshop
$     --copyright %s       a copyright field
```

See this ocio-dev thread for additional usage discussions.

When exporting an ICC Profile, you will be asked to specify your monitor's profile (it will be selected for you by default). This is because ICC Profile are not LUTs per se. An ICC Profile describes a color space and then needs a destination profile to calculate the transformation. So if you have an operation working and looking good on the monitor you're using (and maybe its should choose its profile instead.

5.4.6 FAQ

Can you query a color space by name (like "Rec709") and get back XYZ coordinates of its primaries and whitepoint?

Not currently.

OCIO is a color configuration 'playback' tool that tries to be as flexible as possible; color information such as this is often only needed / relevant at configuration authoring time. Making primaries / whitepoint required would limit the pipeline OCIO could service. In the strictest sense, we would consider OCIO to be a 'baked' representation of color processes, similar to how Alembic files do not store animation rig data, but rather only the baked geometry.

Also, remember that not all colorspaces using in visual effects even have strongly defined color space definitions. For example, scanned film negatives, when linearized with 1d transfer curves (the historical norm in vfx), do not have defined primaries/white point. Each rgb value could of course individually be tied to a specific color, but if you were to do a sweep of the pure 'red channel', for example, you'd find that it creates a curves in chromaticity space, not a single point. (This is due to the 1d linearization not attempting to undo the subtractive processes that created the pixels in the first place.

But many color spaces in OCIO *do* have very strongly defined white points/chromaticities. On the display side, for example, we have very precise information on this.

Perhaps OCIO should include optional metadata to tag outputs? We are looking at this as as a OCIO 1.2 feature.

Can you convert XYZ <-> named color space RGB values?

OCIO includes a MatrixTransform, so the processing capability is there. But there is no convenience function to compute this matrix for you. (We do include other Matrix convenience functions though, so it already has a place to be added. See MatrixTransform in export/OpenColorTransforms.h)

There's talk of extended OCIO 1.2 to have a plugin api where colorspaces could be dynamically added at runtime (such as after reading exr chromaticity header metadata). This would necessitate adding such a feature.

What are the differences between Nuke's Vectorfield and OCIOFileTransform?

(All tests done with Nuke 6.3)

Ext	Details	Notes
3dl	Matched Results	
ccc	n/a	
cc	n/a	
csp	*Difference*	Gain error. Believe OCIO is correct, but need to verify.
cub	Matched Results	Note: Nuke's .cub exporter is broken (gain error)
cube	Matched Results	
hdl	n/a	
mga/m3d	n/a	
spi1d	n/a	
spi3d	n/a	
spimtx	n/a	
vf	*Difference*	Gain error. Believe OCIO is correct, but need to verify.

All gain differences are due to a common 'gotcha' when interpolating 3d luts, related to internal index computation. If you have a 32x32x32 3dlut, when sampling values from (0,1) do you internally scale by 31.0 or 32.0? This is typically well-defined for each format, (in this case the answer is usually 31.0) but when incorrectly handled in an application, you occasionally see gain errors that differ by this amount. (In the case of a 32-sized 3dlut, 32/31 = ~3% error)

What do ColorSpace::setAllocation() and ColorSpace::setAllocationVars() do?

These hints only come into play during LUT baking, and are used to determine proper colorspace allocation handling for 3D LUTs. See this page *How to Configure ColorSpace Allocation* for further information. Note: OCIO v1 used allocation variables for the GPU renderer, but that is no longer necessary for OCIO v2.

5.4.7 Authoring Configurations

5.4.8 Config syntax

OpenColorIO is primarily controlled by a central configuration file, usually named `config.ocio`. This page will only describe how to *syntactically* write this OCIO config - e.g. what transforms are available, or what sections are optional.

This page alone will not help you to write a useful config file! See the *Configurations* section for examples of complete, practical configs, and discussion of how they fit within a facilities workflow.

Please note that you should use the OCIO Python or C++ API to generate the config.ocio file rather than writing the YAML by hand in a text editor. However, if you do ever modify YAML by hand rather than via the API, you should run *ociocheck* on it to ensure that the syntax is correct.

YAML basics

This config file is a YAML document, so it is important to have some basic knowledge of this format.

The Wikipedia article on YAML has a good overview.

OCIO configs typically use a small subset of YAML, so looking at existing configs is probably the quickest way to familiarise yourself (just remember the indentation is important!).

Checking for errors

Use the `ociocheck` command line tool to validate your config. It will inform you of YAML-syntax errors, but more importantly it performs various OCIO-specific validations. There are also validate methods on the Config class in the C++ and Python APIs, (although they do not do the role checking that ociocheck does).

For more information, see the overview of *ociocheck*

Config sections

An OCIO config has the following sections:

- config-header – The header contains the version and LUT search path.
- config-environment – The environment defines the context variables used in the config.
- config-roles – The roles define which color spaces should be used for common tasks.
- config-rules – These rules define sensible defaults that help applications provide a better user experience.
- config-displays-views – This section defines how color spaces should be viewed.
- config-looks – Looks are transforms used to adjust colors, such as to apply a creative effect.
- config-colorspaces – This section defines the scene-referred color space encodings available within the config.
- config-display-colorspaces – This section defines the display-referred color space encodings available within the config.
- config-named-transforms – Named Transforms are a way to provide transforms that do not have a fixed relationship to a specific reference space, such as a utility curve.

A collection of config-transforms is provided for use in the various sections of the config file.

Config header

ocio_profile_version

Required.

By convention, the profile starts with `ocio_profile_version`.

This is a string, specifying which version of the OCIO config syntax is used.

The currently supported version strings are `1` and `2`.

```
ocio_profile_version: 2
```

description

Optional. A brief description of the configuration.

```
description: This is the OCIO config for show: foo.
```

name

Optional. A unique name for the config. Future versions of OCIO might use this as a sort of "namespace" for the color spaces defined in the rest of the config.

```
name: studio-config-v1.0.0_aces-v1.3_ocio-v2.1
```

search_path

Optional. Default is an empty search path.

`search_path` is a colon-separated list of directories. Each directory is checked in order to locate a file (e.g. a LUT).

This works is very similar to how the UNIX `$PATH` env-var works for executables.

A common directory structure for a config is:

```
config.ocio
luts/
  lg10_to_lnf.spi1d
  lg10_to_p3.3dl
```

For this, we would set `search_path` as follows:

```
search_path: "luts"
```

In a color space definition, we might have a FileTransform which refers to the LUT `lg10_to_lnf.spi1d`. It will look in the `luts` directory, relative to the `config.ocio` file's location.

Paths can be relative (to the directory containing `config.ocio`), or absolute (e.g. `/mnt/path/to/my/luts`)

Multiple paths can be specified, including a mix of relative and absolute paths. Each path is separated with a colon `:`

```
search_path: "/mnt/path/to/my/luts:luts"
```

Paths may also be written on separate lines (this is more Windows friendly):

```
search_path:
  - luts1
  - luts2
```

Finally, paths can reference OCIO's context variables:

```
search_path: "/shots/show/$SHOT/cc/data:luts"
```

This allows for some clever setups, for example per-shot LUT's with fallbacks to a default. For more information, see the examples in userguide-looks

family_separator

Optional. Defines the character used to split color space family strings into hierarchical menus. It may only be a single character. If no separator is defined, the Menu Helpers API will not generate hierarchical menus.

```
family_separator: /
```

inactive_colorspaces

Optional. Identify a list of color spaces that should not be used. These spaces may stay in the config and will still work in ColorSpaceTransforms, but they will not be added to application menus. This will be overridden by the environment variable *OCIO_INACTIVE_COLORSPACES*.

```
inactive_colorspaces: [ do_not_use_this_colorspace, prev_version_colorspace ]
```

luma

Deprecated. Optional. Default is the Rec.709 primaries specified by the ASC:

```
luma: [0.2126, 0.7152, 0.0722]
```

These are the luminance coefficients, which can be used by OCIO-supporting applications when adjusting saturation (e.g. in an image-viewer when displaying a single channel)

Note: While the API method is not yet officially deprecated, `luma` is a legacy option from Imageworks' internal, closed-source predecessor to OCIO.

The `luma` value is not respected anywhere within the OCIO library. Also very few (if any) applications supporting OCIO will respect the value either.

Environment

environment

Optional. The envrionment section declares all of the context variables used in the configuration.

```
environment:
  SEQ: default_sequence
  SHOT: $SHOT
```

It is highly recommended that config authors using context variables include the environment section for the following reasons:

- It provides performance benefits to applications
- It will make the config easier to read and maintain
- It allows defining default values
- It improves the validation that may be performed on a config

This config uses two context variables: SEQ and SHOT. SEQ has a default value of default_sequence. This is the value that will be used if the environment does not contain the SEQ variable and the context variable is not otherwise defined. The SHOT variable does not have a default and hence the use of the syntax shown.

The environment must be self-contained and may not refer to any other variables. For instance, in the example above it would not be legal to have `SHOT: $FOO` since FOO is not one of the declared variables.

Every context variable used in the config must be declared since no other environment variables will be loaded into the context. In studios that use a large number of environment variables, this may provide a performance benefit for applications.

Roles

roles

Required.

A "role" is an alternate name to a color space, which can be used by applications to perform task-specific color transforms without requiring the user to select a color space by name.

For example, the Nuke node OCIOLogConvert: instead of requiring the user to select the appropriate log color space, the node performs a transform between `scene_linear` and `compositing_log`, and the OCIO config specifies the project-appropriate color spaces. This simplifies life for artists, as they don't have to remember which is the correct log color space for the current project - the OCIOLogConvert always does the correct thing.

A typical role definition looks like this, taken from the *spi-vfx* example configuration:

```
roles:
  color_picking: cpf
  color_timing: lg10
  compositing_log: lgf
  data: ncf
  default: ncf
  matte_paint: vd8
  reference: lnf
  scene_linear: lnf
  texture_paint: dt16
```

All values in this example (such as `cpf`, `lg10` and `ncf`) refer to color spaces defined later the config, in the `colorspaces` section.

Here is a description of the roles defined within OpenColorIO. Note that application developers may also define roles for config authors to use to control other types of tasks not listed below.

> **Warning:** Unfortunately there is a fair amount of variation in how applications interpret OCIO roles. This section should be expanded to try and clarify the intended usage.

- `aces_interchange` - defines the color space in the config that implements the ACES2065-1 color space defined in SMPTE ST2065-1. This role is used to convert scene-referred color spaces between different configs that both define this role.
- `cie_xyz_d65_interchange` - defines the color space in the config that implements standard CIE XYZ colorimetry, adapted to a D65 white. This role is used to convert display-referred color spaces between different configs that both define this role.

- `color_picking` - colors in a color-selection UI can be displayed in this space, while selecting colors in a different working space (e.g. `scene_linear` or `texture_paint`).
- `color_timing` - color space used for applying color corrections, e.g. user-specified grade within an image viewer (if the application uses the `DisplayTransform::setDisplayCC` API method)
- `compositing_log` - a log color space used for certain processing operations (plate resizing, pulling keys, de-grain, etc). Used by the OCIOLogConvert Nuke node.
- `data` - used when writing data outputs such as normals, depth data, and other "non color" data. The color space in this role should typically have `data: true` specified, so no color transforms are applied.
- `default` - when `strictparsing: false`, this color space is used as a fallback.
- `matte_paint` - color space which matte-paintings are created in (for more information, *see the guide on baking ICC profiles for Photoshop*, and *spi-vfx*).
- `reference` - the color space against which the other color spaces are defined.

Note: The reference role has sometimes been misinterpreted as being the space in which "reference art" is stored in.

- `scene_linear` - the scene-referred linear-to-light color space, often the same as the reference space (see:ref:*faq-terminology*).
- `texture_paint` - similar to `matte_paint` but for painting textures for 3D objects (see the description of texture painting in *SPI's pipeline*).

File & Viewing rules

`file_rules`

Warning: Either the file_rules section or the default role are Required for configs of version 2 or higher.

Use the File Rules to assign a default color space to files based on their path.

Here is example showing the various types of rules that may be defined:

```
file_rules:
  - !<Rule> {name: LogC, extension: "*", pattern: "*LogC*", colorspace: ARRI LogC}
  - !<Rule> {name: OpenEXR, extension: "exr", pattern: "*", colorspace: ACEScg}
  - !<Rule> {name: TIFF, regex: ".*\\.TIF?F", colorspace: sRGB}
  - !<Rule> {name: ColorSpaceNamePathSearch}
  - !<Rule> {name: Default, colorspace: default}
```

The File Rules are a set of mappings that are evaluated from the top down. The first rule to match is what determines which ColorSpace is returned.

There are four types of rules available. Each rule type has a name key that may be used by applications to refer to that rule. Name values must be unique. The other keys depend on the rule type.

1. Basic Rules – This is the basic rule type that uses Unix glob style pattern matching and is thus very easy to use. It contains the keys:

- `name`: Name of the rule.

- `pattern`: Glob pattern to be used for the main part of the name/path. This is case-sensitive. It must be in double-quotes. Set it to "*" if you only want the rule to consider the extension.
- `extension`: Glob pattern or string to be used for the file extension. Note that if glob tokens are not used, the extension will be used in a non-case-sensitive way by default. For example the simple string "exr" would match "exr" and "EXR". If you only want to match "exr", use the glob pattern "[e][x][r]". It must be in double-quotes. Set it to "*" if you only want the rule to consider the pattern.
- `colorspace`: ColorSpace name to be returned.

2. RegEx Rules – This is similar to the basic rule but allows additional capabilities for power-users. It contains the keys:

- `name`: Name of the rule.
- `regex`: Regular expression to be evaluated. It must be in double-quotes.
- `colorspace`: ColorSpace name to be returned.

Note that a backslash character in a RegEx expression needs to be doubled up as \\ (as shown in the example above for the TIFF rule) to make it through the Yaml parsing.

3. OCIO v1 style Rule – This rule allows the use of the OCIO v1 style, where the string is searched for ColorSpace names from the config. This rule may occur 0 or 1 times in the list. The position in the list prioritizes it with respect to the other rules. It has the key:

- `name`: Must be `ColorSpaceNamePathSearch`.

4. Default Rule – The file_rules section must always end with this rule. If no prior rules match, this rule specifies the ColorSpace applications will use. It has the keys:

- `name`: must be `Default`.
- `colorspace`: ColorSpace name to be returned.

Note: OCIO v1 defined a `default` role intended to specify a default color space when reading files. However, given the confusion over the usage of this role, the Default file rule is now the preferred way to indicate this mapping. However, if the file_rules section is not present in the config, the new API will try to use the default role in place of the Default Rule. If both are present, the Default Rule takes precedence. If both the file_rules section and the default role are missing, an exception will be thrown when loading the config.

Note that the strictparsing token does not affect the behavior of the File Rules API. In other words, evaluating the rules will always result in a ColorSpace being available to an application. However, the API also allows the application to know which rule was the matching one. So apps that want to work in "strict" mode should first check if strictparsing is true and if so check to see if the matching rule was the Default Rule. If so, it could then notify the user and take whatever action is appropriate. (As an alternative to checking which rule number was matched, the API call `filepathOnlyMatchesDefaultRule` may be used instead.)

Roles may be used rather than ColorSpace names in the rules.

It is also legal for rules to have additional key:value pairs where the value may be an arbitrary string. The API provides access to getting/setting these additional pairs and will preserve them on a Config read/write. These may be used to define application-specific behavior.

Note to developers: The older `parseColorSpaceFromString` API call is now deprecated and should be replaced with `getColorSpaceFromFilepath`.

strictparsing

Optional. Valid values are `true` and `false`. Default is `true` (assuming a config is present):

```
strictparsing: true
```

> **Warning:** This attribute is from OCIO v1. In OCIO v2, the FileRules system was introduced and so the `strictparsing` attribute is less relevant now. The `parseColorSpaceFromString` API call is now deprecated and the proper way to obtain this information is `getColorSpaceFromFilepath`. The FileRules always return a default color space but the API `filepathOnlyMatchesDefaultRule` may be used by applications that want to take some special action if `strictparsing` is true.

OCIO v1 provided a mechanism for applications to extract the colorspace from a filename (the `parseColorSpaceFromString` API method).

So for a file like `example_render_v001_lnf.0001.exr` it will determine the colorspace `lnf` (it being the right-most substring containing a colorspace name).

However, if the colorspace cannot be determined and `strictparsing: true`, it will return an empty string.

If the colorspace cannot be determined and `strictparsing: false`, the default role will be used. This allows unhandled images to operate in "non-color managed" mode.

Application authors should note: when no config is present (e.g. via `$OCIO`), the default internal profile specifies `strictparsing=false`, and the default color space role is `raw`. This means that ANY string passed to OCIO will be parsed as the default `raw`. This is nice because in the absence of a config, the behavior from your application perspective is that the library essentially falls back to "non-color managed".

viewing_rules

Optional.

Use the Viewing Rules to assign default views to color spaces.

The Viewing Rules allow config authors to help applications provide a better user experience by specifying the views appropriate to use for a given color space. For example, applications may use the default view when making thumbnail images for its user interface. Likewise, an application could select the default view the first time it displays an image in a viewport.

Here is an example:

```
viewing_rules:
  - !<Rule> {name: video-spaces, colorspaces: [sRGB, Rec709]}
  - !<Rule> {name: data-spaces, colorspaces: [alpha, normals]}

displays:
  sRGB:
    - !<View> {name: Video, view_transform: colorimetry, display_colorspace: sRGB, rule:␣
↪video-spaces}
    - !<View> {name: Raw, colorspace: nc10, rule: data-spaces}
```

This is helpful for situations where a given view is intended for use with just a few specific color spaces. However in other situations, it would be helpful to be able to define rules to be used with a broader set of color spaces.

Color spaces may now have an "encoding" attribute to allow grouping color spaces into groups such as "scene-linear", "log", "sdr-video", and "data". The Viewing Rules makes it possible to define a rule based on the encoding attribute of a color space rather than a set of named color spaces. For example:

```
viewing_rules:
  - !<Rule> {name: scene-linear-or-log, encodings: [scene-linear, log]}

displays:
  sRGB:
    - !<View> {name: ACES, view_transform: ACES-sdr-video, display_colorspace: sRGB,␣
↪rule: scene-linear-or-log}
```

The colorspaces and encodings attributes may contain a single value or a list of values. It is illegal for a rule to define both a list of colorspaces and encodings simultaneously.

Also, similar to the file_rules, it is allowed for a rule to define a set of custom key/value pairs like this:

```
- !<Rule> { name: scene-linear-rule, encodings: scene-linear, custom: {key1: value1,␣
↪key2: value2} }
```

The key names and values are arbitrary strings. This may be useful to control application-specific behavior.

A Viewing Rule may contain the following keys:

- `name`: The name of the rule (must be unique).
- `encodings`: The color space encodings used by the rule (may be a list).
- `colorspaces`: The color space names used by the rule (may be a list).
- `custom`: A set of arbitrary key/value string pairs.

The API allows applications to request the list of views for a given color space. This uses the viewing rules to filter the views for the given display based on the color space name and encoding. Views that do not have a rules attribute are always returned (so if no rules are present, the results are the same as the unfiltered API call).

Note that the active_views may be used to remove views that are not appropriate for a given user or workstation. If the active_views list is non-empty, any views that are not in that list will not appear in the results provided to the application (regardless of whether the view appears in a rule).

Furthermore, active_views will continue to sort (that is, determine the index order) the list of views in all of the API calls.

The first allowed view for a color space is the default.

Displays & Views

displays

Required.

This section defines all the display devices which will be used. For example you might have a sRGB display device for artist workstations, a DCIP3 display device for the screening room projector.

Each display device has a number of "views". These views provide different ways to display the image on the selected display device. Examples of common views are:

- "Film" to emulate the final projected result on the current display

- "Log" to send log-space pixel values directly to the display, resulting in a "lifted" image useful for checking black-levels.
- "Raw" when assigned a colorspace with `raw: yes` set will show the unaltered image, useful for tech-checking images

An example of the `displays` section from the *spi-vfx* config:

```
displays:
  DCIP3:
    - !<View> {name: Film, colorspace: p3dci8}
    - !<View> {name: Log, colorspace: lg10}
    - !<View> {name: Raw, colorspace: nc10}
  sRGB:
    - !<View> {name: Film, colorspace: srgb8}
    - !<View> {name: Log, colorspace: lg10}
    - !<View> {name: Raw, colorspace: nc10}
    - !<View> {name: Film, colorspace: srgb8}
```

All the colorspaces (`p3dci8`, `srgb8` etc) refer to colorspaces defined later in the config.

Unless the `active_displays` and `active_views` sections are defined, the first display and first view will be the default.

A view may be defined as a transform directly from the scene-referred reference space to the display, as illustrated above. Alternatively a view may be defined using the combination of a View Transform and a Display Color Space. In this case, the View Transform converts from the scene-referred reference space to the display-referred reference space and then the Display Color Space is used to convert from the display-referred reference space to the display. Here is an example:

```
displays:
  DCIP3:
    - !<View> {name: Film, view_transform: FilmView, display_colorspace: DCIP3}

view_transforms:
  - !<ViewTransform>
    name: FilmView
    from_reference: <omitted for brevity>

display_colorspaces:
  - !<ColorSpace>
    name: DCIP3
    from_display_reference: <omitted for brevity>
```

The keys allowed with a View are:

- `name`: A name for the View.
- `colorspace`: The color space whose transform will be used to convert from the reference space to the display.
- `view_transform`: The transform used to convert from either the scene-referred reference to the display-referred reference space or as a transform to be applied in the display-referred reference.
- `display_colorspace`: The display color space that is used to convert from the display-referred reference space to the display.
- `looks`: One or more Look Transforms to be applied. The string may use the '+' character to apply a look in the forward direction and the '-' character to apply in reverse. See config-looks

- `rule`: The viewing rule to be used with this View. See config-viewing-rules
- `description`: A description string for this View.

Note that a View may use either the colorspace key or it may use both the view_transform and dispay_colorspace keys. No other combinations are allowed.

view_transforms

Optional. Defines transforms to convert between the OCIO reference spaces.

An OCIO config may contain two Reference spaces: one scene-referred and the other display-referred. The View Transforms are mappings between these reference spaces. There are a variety of terms used in the industry for these mappings. In ISO 22028-1 and in ACES, they are called "color-rendering transforms" and in the ITU standards for HDR television (such as ITU-R BT.2100 and BT.2390) they are called an "Opto-optical Transfer Function" or OOTF.

It is also possible to specify a View Transform for adjusting display-referred reference space values. In other words, it may convert from the display-referred reference space back to itself. This is useful for describing transforms used for HDR to SDR video conversion, or vice versa.

A View Transform may use the following keys:

- `name`: A name for the ViewTransform.
- `description`: A description of the ViewTransform.
- `family`: A family string (similar to ColorSpace).
- `categories`: The categories used for menu filtering (similar to ColorSpace).
- `amf_transform_ids`: The ACES transform IDs to be used with AMF files. Note this is a sub-key under the `interchange` key.
- `from_scene_reference`: The transform from the scene-referred reference space to the display-referred reference space.
- `to_scene_reference`: The transform from the display-referred reference space to the scene-referred reference space.
- `from_display_reference`: The transform from the display-referred reference space to the display-referred reference space.
- `to_display_reference`: The inverse of the from_display_reference transform.

default_view_transform

Optional. Defines the default view transform.

The default view transform is the view transform that is used if a ColorSpaceTransform needs to convert between a scene-referred and display-referred colorspace. If this key is missing, the first view transform in the config is used.

Please note that this is *not* the default view for the config. The default view is the first view in the `active_views` list and should be obtained using the `getDefaultView` API function.

```
default_view_transform: un-tone-mapped
```

shared_views

Optional. Allows a view to be defined once for use in multiple displays.

For example, this:

```
shared_views:
  - !<View> {name: Log, colorspace: lg10}
  - !<View> {name: Raw, colorspace: nc10}

displays:
  DCIP3:
    - !<View> {name: Film, colorspace: p3dci8}
    - !<Views> [ Log, Raw ]
  sRGB:
    - !<View> {name: Film, colorspace: srgb8}
    - !<Views> [ Log, Raw ]
```

Is equivalent to this:

```
displays:
  DCIP3:
    - !<View> {name: Film, colorspace: p3dci8}
    - !<View> {name: Log, colorspace: lg10}
    - !<View> {name: Raw, colorspace: nc10}
  sRGB:
    - !<View> {name: Film, colorspace: srgb8}
    - !<View> {name: Log, colorspace: lg10}
    - !<View> {name: Raw, colorspace: nc10}
```

A shared view may use the special token <USE_DISPLAY_NAME> to request that the display_colorspace name be set equal to the display name that the shared view is used with. In the following example, when the ACES shared view is used with the sRGB display, its display_colorspace is set to "sRGB" and when it is used with the Rec709 display, its display_colorspace is set to "Rec709".

```
shared_views:
  - !<View> {name: ACES, view_transform: ACES-sdr-video,
             display_colorspace: <USE_DISPLAY_NAME>}
  - !<View> {name: Log, colorspace: lg10}
  - !<View> {name: Raw, colorspace: nc10}

displays:
  sRGB:
    - !<Views> [ ACES, Log, Raw ]
  Rec709:
    - !<Views> [ ACES, Log, Raw ]

display_colorspaces:
  - !<ColorSpace>
    name: sRGB
    from_display_reference: <omitted for brevity>
  - !<ColorSpace>
    name: Rec709
    from_display_reference: <omitted for brevity>
```

As with any other view, a shared view may appear (or not) in the list of active_views.

Application developers do not need to worry about shared views since the API presents them as if they were a typical View.

active_displays

Optional. Default is for all displays to be visible, and to respect order of items in `displays` section.

You can choose what display devices to make visible in UI's, and change the order in which they appear.

Given the example `displays` block in the previous section - to make the sRGB device appear first:

```
active_displays: [sRGB, DCIP3]
```

To display only the `DCIP3` device, simply remove `sRGB`:

```
active_displays: [DCIP3]
```

The value can be overridden by the `OCIO_ACTIVE_DISPLAYS` env-var. This allows you to make the `sRGB` the only active display, like so:

```
active_displays: [sRGB]
```

Then on a review machine with a DCI P3 projector, set the following environment variable, making `DCIP3` the only visible display device:

```
export OCIO_ACTIVE_DISPLAYS="DCIP3"
```

Or specify multiple active displays, by separating each with a colon:

```
export OCIO_ACTIVE_DISPLAYS="DCIP3:sRGB"
```

active_views

Optional. Default is for all views to be visible, and to respect order of the views under the display.

Works identically to `active_displays`, but controls which *views* are visible.

Overridden by the `OCIO_ACTIVE_VIEWS` env-var:

```
export OCIO_ACTIVE_VIEWS="Film:Log:Raw"
```

virtual_display

Optional. A virtual display may be defined to allow OCIO to instantiate new displays from ICC profiles.

The syntax is similar to a conventional display. It may incorporate both views and shared views. There may only be one virtual display in a config.

When OCIO instantiates a display from an ICC monitor profile it will create a display colorspace which is used with any views that have the display_colorspace set to <USE_DISPLAY_NAME>.

So in this example, if the application asks OCIO to instantiate a display from an ICC profile, the user would then see a second display, in addition to "sRGB", named after the ICC profile. The views available for that new display are taken from the virtual display.

```
displays:
  sRGB:
    - !<View> {name: ACES, view_transform: ACES-sdr-video,
               display_colorspace: <USE_DISPLAY_NAME>}
    - !<View> {name: Log, colorspace: lg10}
    - !<Views> [ Raw ]

virtual_display:
  - !<View> {name: ACES, view_transform: ACES-sdr-video,
             display_colorspace: <USE_DISPLAY_NAME>}
  - !<View> {name: Log, colorspace: lg10}
  - !<Views> [ Raw ]
```

Looks

looks

Optional.

This section defines a list of "looks". A look is a color transform defined similarly to a colorspace, with a few important differences.

For example, a look could be defined for a "first pass DI beauty grade", which is used to view shots with a rough approximation of the final grade.

When the look is defined in the config, you must specify a name, the color transform, and the colorspace in which the grade is performed (the "process space"). You can optionally specify an inverse transform for when the look transform is not trivially invertable (e.g. it applies a 3D LUT)

When an application applies a look, OCIO ensures the grade is applied in the correct colorspace (by converting from the input colorspace to the process space, applies the look's transform, and converts the image to the output colorspace)

Here is a simple `looks:` section, which defines three looks:

```
looks:
  - !<Look>
    name: beauty
    process_space: lnf
    transform: !<CDLTransform> {slope: [1, 2, 1]}

  - !<Look>
    name: neutral
    process_space: lg10
    transform: !<FileTransform> {src: 'neutral-${SHOT}-${SEQ}.csp', interpolation:␣
↪linear }
    inverse_transform: !<FileTransform> {src: 'neutral-${SHOT}-${SEQ}-reverse.csp',␣
↪interpolation: linear }

  - !<Look>
    name: ACES 1.3 Reference Gamut Compression
    process_space: ACES2065-1
    description: LMT (applied in ACES2065-1) to compress scene-referred values from␣
↪common cameras into the AP1 gamut
    interchange:
```

(continues on next page)

(continued from previous page)

```
      amf_transform_ids: |
        urn:ampas:aces:transformId:v2.0:InvLook.Academy.ReferenceGamutCompress.a2.v1
        urn:ampas:aces:transformId:v2.0:Look.Academy.ReferenceGamutCompress.a2.v1
    transform: !<BuiltinTransform> {style: ACES-LMT - ACES 1.3 Reference Gamut␣
↪Compression
```

Here, the "beauty" look applies a simple, static ASC CDL grade, making the image very green (for some artistic reason!). The beauty look is applied in the scene-linear `lnf` colorspace (this colorspace is defined elsewhere in the config.

Next is a definition for a "neutral" look, which applies a shot-specific CSP LUT, dynamically finding the correct LUT based on the SEQ and SHOT context variables.

For example if `SEQ=ab` and `SHOT=1234`, this look will search for a LUT named `neutral-ab-1234.csp` in locations specified in `search_path`.

The `process_space` here is `lg10`. This means when the look is applied, OCIO will perform the following steps:

- Transform the image from it's current colorspace to the `lg10` process space
- Apply apply the FileTransform (which applies the grade LUT)
- Transform the graded image from the process space to the output colorspace

The "beauty" look specifies the optional `inverse_transform`, because in this example the neutral CSP files contain a 3D LUT. For many transforms, OCIO will automatically calculate the inverse transform (as with the "beauty" look), however with a 3D LUT the inverse transform needs to be defined.

If the look was applied in reverse, and `inverse_transform` as not specified, then OCIO would give a helpful error message. This is commonly done for non-invertable looks

As in colorspace definitions, the transform can be specified as a series of transforms using the `GroupTransform`, for example:

```
looks:
  - !<Look>
    name: beauty
    process_space: lnf
    transform: !<GroupTransform>
      children:
        - !<CDLTransform> {slope: [1, 2, 1]}
        - !<FileTransform> {src: beauty.spi1d, interpolation: nearest}
```

Colorspaces

colorspaces

Required.

This section is a list of the scene-referred colorspaces in the config. A colorspace may be referred to elsewhere within the config (including other colorspace definitions), and are used within OCIO-supporting applications.

A colorspace may use the following keys:

to_scene_reference and from_scene_reference

These keys specify the transforms that define the relationship between the colorspace and the scene-referred reference space.

Note: In OCIO v1, the keys `to_reference` and `from_reference` were used (since there was only one reference space). These are still supported as synonyms.

Here is a example of a very simple `colorspaces` section, modified from the *spi-vfx* example config:

```
colorspaces:
  - !<ColorSpace>
    name: lnf
    bitdepth: 32f
    description: |
      lnf : linear show space

  - !<ColorSpace>
    name: lg16
    bitdepth: 16ui
    description: |
      lg16 : conversion from film log
    to_scene_reference: !<FileTransform> {src: lg16_to_lnf.spi1d, interpolation: nearest}
```

First the `lnf` colorspace (short for linear float) is used as our reference colorspace. The name can be anything, but the idea of a reference colorspace is an important convention within OCIO: **all other colorspaces are defined as transforms either to or from this colorspace.**

The `lg16` colorspace is a 16-bit log colorspace (see *spi-vfx* for an explanation of this colorspace). It has a name, a bit-depth and a description.

The `lg16` colorspace is defined as a transform from `lg16` to the reference colorspace (`lnf`). That transform is to apply the LUT `lg16_to_lnf.spi1d`. This LUT has an input of `lg16` integers and outputs linear 32-bit float values

Since the 1D LUT is automatically invertable by OCIO, we can use this colorspace both to convert `lg16` images to `lnf`, and `lnf` images to `lg16`

Importantly, because of the reference colorspace concept, we can convert images from `lg16` to the reference colorspace, and then on to any other colorspace.

Here is another example colorspace, which is defined using `from_scene_reference`.

```
- !<ColorSpace>
  name: srgb8
  bitdepth: 8ui
  description: |
    srgb8 :rgb display space for the srgb standard.
  from_scene_reference: !<FileTransform> {src: srgb8.spi3d, interpolation: linear}
```

We use `from_scene_reference` here because we have a LUT which transforms from the reference colorspace (`lnf` in this example) to sRGB.

In this case `srgb8.spi3d` is a complex 3D LUT which cannot be inverted, so it is considered a "display only" colorspace. If we did have a second 3D LUT to apply the inverse transform, we can specify both `to_scene_reference` and `from_scene_reference`

```
- !<ColorSpace>
  name: srgb8
  bitdepth: 8ui
  description: |
    srgb8 :rgb display space for the srgb standard.
  from_scene_reference: !<FileTransform> {src: lnf_to_srgb8.spi3d, interpolation: linear}
  to_scene_reference: !<FileTransform> {src: srgb8_to_lnf.spi3d, interpolation: linear}
```

Using multiple transforms

The previous example colorspaces all used a single transform each, however it is often useful to use multiple transforms to define a colorspace.

```
- !<ColorSpace>
  name: srgb8
  bitdepth: 8ui
  description: |
    srgb8 :rgb display space for the srgb standard.
  from_scene_reference: !<GroupTransform>
    children:
      - !<ColorSpaceTransform> {src: lnf, dst: lg16}
      - !<FileTransform> {src: lg16_to_srgb8.spi3d, interpolation: linear}
```

Here to get from the reference colorspace, we first use the `ColorSpaceTransform` to convert from `lnf` to `lg16`, then apply our 3D LUT on the log-encoded images.

This primarily demonstrates the meta-transform `GroupTransform`: a transform which simply composes two or more transforms together into one. Anything that accepts a transform like `FileTransform` or `CDLTransform` will also accept a `GroupTransform`

It is also worth noting the `ColorSpaceTransform`, which transforms between `lnf` and `lg16` colorspaces (which are defined within the current config).

Example transform steps

This section explains how OCIO internally applies all the transforms. It can be skipped over if you understand how the reference colorspace works.

```
colorspaces:
  - !<ColorSpace>
    name: lnf
    bitdepth: 32f
    description: |
      lnf : linear show space

  - !<ColorSpace>
    name: lg16
    bitdepth: 16ui
    description: |
      lg16 : conversion from film log
    to_scene_reference: !<FileTransform> {src: lg16.spi1d, interpolation: nearest}
```

(continues on next page)

(continued from previous page)

```
- !<ColorSpace>
  name: srgb8
  bitdepth: 8ui
  description: |
    srgb8 :rgb display space for the srgb standard.
  from_scene_reference: !<GroupTransform>
    children:
      - !<ColorSpaceTransform> {src: lnf, dst: lg16}
      - !<FileTransform> {src: lg16_to_srgb8.spi3d, interpolation: linear}
```

To explain how this all ties together to display an image, say we have an image in the `lnf` colorspace (e.g. a linear EXR) and wish to convert it to `srgb8` - the transform steps are:

- `ColorSpaceTransform` is applied, converting from lnf to lg16
- The `FileTransform` is applied, converting from lg16 to srgb8.

A more complex example: we have an image in the `lg16` colorspace, and convert to `srgb8` (using the lg16 definition from earlier, or the *spi-vfx* config):

First OCIO converts from lg16 to the reference space, using the transform defined in lg16's to_scene_reference:

- `FileTransform` applies the lg16.spi1d

With the image now in the reference space, srgb8's transform is applied:

- ColorSpaceTransform to transform from lnf to lg16
- FileTransform applies the `lg16_to_srgb8.spi3d` LUT.

Note: OCIO has an transform optimizer which removes redundant steps, and combines similar transforms into one operation.

In the previous example, the complete transform chain would be "lg16 -> lnf, lnf -> lg16, lg16 -> srgb8". However the optimizer will reduce this to "lg16 -> srgb".

---

encoding

Optional. Specify how color space values are numerically encoded.

It is very helpful for applications to be able to know the basic type of encoding of a color space. For example, it is well known that the performance of various types of image processing algorithms varies based on the type of encoding. Applying a spatial filter to a scene-linear image gives a different subjective result than if applied to the same image encoded in a log color space. Likewise certain algorithms such as keying or tracking may assume that the color encoding is roughly perceptually uniform and thus may have difficulties with scene-linear images.

The allowed values and definitions are:

`scene-linear` – A scene-referred encoding where the numeric representation is proportional to scene luminance. Examples: ACES2065-1, ACEScg.

`display-linear` – A display-referred encoding where the numeric representation is proportional to display luminance. Example: CIE XYZ values measured off of a display or projection screen.

`log` – A scene-referred encoding where the numeric representation is roughly proportional to the logarithm of scene-luminance (often with some divergence in the shadows as with most camera log encodings). Examples: ACEScct, ACEScc, ARRI LogC, Sony S-Log3/S-Gamut3.

sdr-video – A display-referred encoding where the numeric representation is proportional to an SDR video signal. Examples: Rec.709/Rec.1886 video, sRGB.

hdr-video – A display-referred encoding where the numeric representation is proportional to an HDR video signal. Examples: Rec.2100/PQ or Rec.2100/HLG.

edr-video – A display-referred encoding where the numeric representation is proportional to an SDR video signal but allows extended range values intended for use on extended or high-dynamic range displays. Example: Display P3 HDR.

data – A non-color channel. Note that typically such a color space would also have the isdata attribute set to true. Examples: alpha, normals, Z-depth.

bitdepth

Optional. Default: 32f

Specify an appropriate bit-depth for the colorspace, and applications can use this to automatically output images in the correct bit-depth.

Valid options are:

- 8ui
- 10ui
- 12ui
- 14ui
- 16ui
- 32ui
- 16f
- 32f

The number is in bits. ui stands for unsigned integer. f stands for floating point.

Example:

```
- !<ColorSpace>
  name: srgb8
  bitdepth: 8ui

  from_scene_reference: [...]
```

isdata

Optional. Default: false. Boolean.

The isdata key on a colorspace informs OCIO that this colorspace is used for non-color data channels, such as the "normals" output of a a multipass 3D render.

Here is example "non-color" colorspace from the *spi-vfx* config:

```
- !<ColorSpace>
  name: ncf
  family: nc
  equalitygroup:
  bitdepth: 32f
  description: |
    ncf :nc,Non-color used to store non-color data such as depth or surface normals
  isdata: true
  allocation: uniform
```

equalitygroup

Optional.

If two colorspaces are in the "equality group", transforms between them are considered non-operations.

You might have multiple colorspaces which are identical, but operate at different bit-depths.

For example, see the `lg10` and `lg16` colorspaces in the *spi-vfx* config. If loading a `lg10` image and converting to `lg16`, no transform is required. This is of course faster, but may cause an unexpected increase in precision (e.g. it skip potential clamping caused by a LUT)

```
- !<ColorSpace>
  name: lg16
  equalitygroup: lg
  bitdepth: 16ui
  to_scene_reference: !<FileTransform> {src: lg16.spi1d, interpolation: nearest}

- !<ColorSpace>
  name: lg10
  equalitygroup: lg
  bitdepth: 10ui
  to_scene_reference: !<FileTransform> {src: lg10.spi1d, interpolation: nearest}
```

Do not put different colorspaces in the same equality group. For logical grouping of "similar" colorspaces, use the `family` option.

family

Optional.

Allows for logical grouping of colorspaces within a UI.

For example, a series of "log" colorspaces could be put in one "family". Within a UI like the Nuke `OCIOColorSpace` node, these will be grouped together.

The Menu Helpers API allows applications to build hierarchical menus for color spaces based on the `family` key. The `family_separator` key of the config is used to define the character used to separate the family string into tokens.

```
family_separator: /

color_spaces:
  - !<ColorSpace>
```

(continues on next page)

(continued from previous page)

```
    name: ACME_log4
    family: Log/Cameras/ACME
    equalitygroup: ACME_log4
    [...]

  - !<ColorSpace>
    name: ACEScct
    family: Log/ACES
    equalitygroup: ACEScct
    [...]

  - !<ColorSpace>
    name: Rec.709
    family: Video/Broadcast/SDR
    equalitygroup: Rec.709
    [...]
```

Unlike `equalitygroup`, the `family` has no impact on image processing.

aliases

Optional.

The aliases key is used to define alternate names for the colorspace. For example, it may be useful to define a shorter version of the name that is easier to include in texture path names. Or it may be necessary to define an older version of the name for the color space for backwards compatibility.

```
aliases: [shortName, obsoleteName]
```

interop_id

Optional.

OCIO supports the Color Interop ID developed by the Color Interop Forum. This allows config authors to set an ID on the color spaces in a config to unambiguously identify them as conforming to the recommendations of the ASWF Color Interop Forum. These IDs may then be used in file formats including OpenEXR and OpenUSD.

Note that if a color space has an `interop_id`, that same string must appear as an alias or as the name of a color space in the config. Unlike color space names or aliases, more than one color space may use the same interop ID string. This is because sometimes a config may have multiple color spaces that correspond to a given external standard. In this situation, only one of those color spaces will have the alias and that will be the one that is used by default, for example when an application loads an OpenEXR file that uses that interop ID.

The interop ID is not an arbitrary string, it must adhere to the rules and structure defined by the Color Interop Forum. For example, only certain characters are allowed, and color spaces not published in a Color Interop Forum recommendation must include a namespace prefix.

Although config authors may use a variety of names for a given color space, based on the needs and conventions of their studio, the intent is that everyone will use the same interop IDs and that this will allow better tagging in file formats than the local color space names that are only meaningful within the context of a specific config.

Developers should note that these IDs are for use internally in file formats but the color space's name attribute is still what should be used in a UI.

The interop ID may be used in configs of version 2.0 or higher.

```
- !<ColorSpace>
  name: ACEScg
  aliases: [ACES - ACEScg, lin_ap1, lin_ap1_scene]
  interop_id: lin_ap1_scene
  [...]
```

allocation and allocationvars

Optional.

These two options were used in OCIO v1 when transforms were applied on the GPU. However, the new GPU renderer in OCIO v2 does not need these.

However, they may still be used to automatically generate a "shaper LUT" when *baking LUT's* unless one is explicitly specified (not all output formats utilise this)

For a detailed description, see *How to Configure ColorSpace Allocation*

Example of a "0-1" colorspace

```
allocation: uniform
allocationvars: [0.0, 1.0]
```

```
allocation: lg2
allocationvars: [-15, 6]
```

description

Optional.

A human-readable description of the colorspace.

The YAML syntax allows for either single-line descriptions:

```
- !<ColorSpace>
  name: kodaklog
  [...]
  description: A concise description of the kodaklog colorspace.
```

Or multiple-lines:

```
- !<ColorSpace>
  name: kodaklog
  [...]
  description:
    This is a multi-line description of the kodaklog colorspace,
    to demonstrate the YAML syntax for doing so.

    Here is the second line. The first one will be unwrapped into
    a single line, as will this one.
```

It's common to use literal | block syntax to preserve all newlines:

```
- !<ColorSpace>
  name: kodaklog
  [...]
  description: |
    This is one line.
    This is the second.
```

interchange

Optional.

The interchange attributes are provided to allow better interop between OCIO and other color management standards.

The `amf_transform_ids` is a newline-separated list of transform IDs intended for use with the ACES Metadata File (AMF). Please note that this should include both the forward and inverse IDs (if available). For display color spaces, this should include the ACES Output Transform IDs used with that display. Note that the same attribute for the View Transforms and Looks sections of the config should be populated as well.

```
- !<ColorSpace>
  name: ACEScct
  [...]
  interchange:
    amf_transform_ids: |
      urn:ampas:aces:transformId:v2.0:CSC.Academy.ACES_to_ACEScct.a2.v1
      urn:ampas:aces:transformId:v2.0:CSC.Academy.ACEScct_to_ACES.a2.v1
```

The `icc_profile_name` is intended to identify an ICC profile to be used in connection with a given color space in the config. For example, some applications may want to embed an ICC profile when writing image files to indicate the color space. The profile should be located somewhere on the config's search path. Developers may use the resolveFileLocation function on the Context class to resolve the full path to the file.

```
- !<ColorSpace>
  name: sRGB - Display
  [...]
  interchange:
    icc_profile_name: srgb_profile.icc
```

The interchange attributes may be used in configs of version 2.5 or higher.

Display Colorspaces

display_colorspaces

Optional.

This section is a list of all the display-referred colorspaces in the config. A display colorspace is very similar to a colorspace except its transforms go from or to the display-referred reference space rather than the scene-referred reference space.

A display colorspace may use all the same keys as a colorspace except it uses `to_display_reference` and `from_display_reference` rather than `to_scene_reference` and `from_scene_reference` to specify its transforms.

```
display_colorspaces:

  - !<ColorSpace>
    name: sRGB
    family:
    description: |
      sRGB monitor (piecewise EOTF)
    isdata: false
    categories: [ file-io ]
    encoding: sdr-video
    from_display_reference: !<GroupTransform>
      children:
        - !<BuiltinTransform> {style: "DISPLAY - CIE-XYZ-D65_to_sRGB"}
        - !<RangeTransform> {min_in_value: 0., min_out_value: 0., max_in_value: 1., max_
↪out_value: 1.}
```

Available transforms

AllocationTransform

Transforms from reference space to the range specified by the `vars:`

Keys:

- `allocation`
- `vars`
- `direction`

BuiltInTransform

Builds one of a known set of transforms, on demand

Keys:

- `style`
- `direction`

CDLTransform

Applies an ASC CDL compliant grade

Keys:

- `slope`
- `offset`
- `power`
- `sat`
- `style`
- `name`

- direction

ColorSpaceTransform

Transforms from `src` colorspace to `dst` colorspace.

Keys:

- src
- dst
- data_bypass
- direction

DisplayViewTransform

Applies a View from one of the displays.

Keys:

- src
- display
- view
- looks_bypass
- data_bypass
- direction

ExponentTransform

Raises pixel values to a given power (often referred to as "gamma")

```
!<ExponentTransform> {value: [1.8, 1.8, 1.8, 1]}
```

Keys:

- value
- style
- name
- direction

ExponentWithLinearTransform

Applies a power but with a linear section near black. May be used to implement sRGB, CIE L*, and the Rec.709 camera OETF (not the display!).

Keys:

- gamma
- offset
- style
- name
- direction

ExposureContrastTransform

Applies an exposure, contrast, or gamma adjustment. Uses dynamic properties for optimal live adjustments (e.g., in viewports).

```
!<ExposureContrastTransform> {style: linear, exposure: -1.5,
  contrast: 0.5, gamma: 1.1, pivot: 0.18}
```

Keys:

- exposure
- contrast
- pivot
- gamma
- style
- log_exposure_step
- log_midway_gray
- name
- direction

FileTransform

Applies a lookup table (LUT)

Keys:

- src
- cccid
- cdl_style
- interpolation
- direction

FixedFunctionTransform

Applies one of a set of fixed, special purpose, mathematical operators.

Keys:

- style
- params
- name
- direction

GradingPrimaryTransform

Applies primary color correction.

Keys:

- style
- brightness
- contrast
- pivot
- offset
- exposure
- lift
- gamma
- gain
- saturation
- clamp
- name
- direction

GradingRGBCurveTransform

Applies a spline-based curve on the red, green, and blue channels.

Keys:

- style
- red
- green
- blue
- master
- lintolog_bypass
- name

- direction

GradingHueCurveTransform

Applies a spline-based curve controlling hue, saturation, and luma.

Keys:

- style
- hue_hue
- hue_sat
- hue_lum
- lum_sat
- sat_sat
- lum_lum
- sat_lum
- hue_fx
- rgb2hsy_bypass
- name
- direction

GradingToneTransform

Applies an adjustment to various tonal ranges.

Keys:

- style
- blacks
- shadows
- midtones
- highlights
- whites
- s_contrast
- name
- direction

GroupTransform

Combines multiple transforms into one.

```
colorspaces:

  - !<ColorSpace>
    name: adx10

    [...]

    to_reference: !<GroupTransform>
      children:
        - !<FileTransform> {src: adx_adx10_to_cdd.spimtx}
        - !<FileTransform> {src: adx_cdd_to_cid.spimtx}
```

A group transform is accepted anywhere a "regular" transform is.

Keys:

- children
- name
- direction

LogAffineTransform

Applies a logarithm as well as a scale and offset on both the linear and log sides. May be used to implement Cineon or Pivoted (Josh Pines) style lin-to-log transforms.

Keys:

- base
- lin_side_offset
- lin_side_slope
- log_side_offset
- log_side_slope
- name
- direction

LogCameraTransform

Similar to LogAffineTransform but also allows a linear section near black. May be used to implement the ACEScct non-linearity as well as many camera vendor lin-to-log transforms.

Keys:

- base
- lin_side_offset
- lin_side_slope

- log_side_offset
- log_side_slope
- lin_side_break
- linear_slope
- name
- direction

LogTransform

Applies a mathematical logarithm with a given base to the pixel values.

Keys:

- base
- name
- direction

LookTransform

Applies a named look

Keys:

- src
- dst
- looks
- direction

MatrixTransform

Applies a matrix transform to the pixel values

Keys:

- matrix
- offset
- name
- direction

RangeTransform

Applies an affine transform (scale & offset) and clamps values to min/max bounds.

Keys:

- min_in_value
- max_in_value
- min_out_value
- max_out_value
- style
- name
- direction

Note: If a min_in_value is present, then min_out_value must also be present and the result is clamped at the low end. Similarly, if max_in_value is present, then max_out_value must also be present and the result is clamped at the high end.

Named Transforms

Sometimes it is helpful to include one or more transforms in a config that are essentially stand-alone transforms that do not have a fixed relationship to a reference space or a process space. An example would be a "utility curve" transform where the intent is to simply apply a LUT1D without any conversion to a reference space. In these cases, a named_transforms section may be added to the config with one or more named transforms.

Note that named transforms do not show up in color space menus by default, so the application developer must implement support to make them available to users.

This feature may be used to emulate older methods of color management that ignored the RGB primaries and simply applied one-dimensional transformations. However, config authors are encouraged to implement transforms as normal OCIO color spaces wherever possible.

Named transforms support the keys:

- name
- aliases
- description
- family
- categories
- encoding
- transform
- inverse_transform

```
named_transforms:
  - !<NamedTransform>
    name: Utility Curve -- Cineon Log to Lin
    transform: !<FileTransform> {src: logtolin_curve.spi1d}
```

5.4.9 Looks Example

Warning: This section is from OCIO v1 and has not been updated yet.

A "look" is a named color transform, intended to modify the look of an image in a "creative" manner (as opposed to a colorspace definition which tends to be technically/mathematically defined). An OCIO look typically exists as a flexible addendum to a defined viewing transform.

Examples of looks may be a neutral grade, to be applied to film scans prior to VFX work, or a per-shot DI grade decided on by the director, to be applied just before the viewing transform.

Looks are defined similarly to colorspaces, you specify a name and a transform (possibly a GroupTransform containing several other transforms), and optionally an inverse transform.

Where looks differ from colorspace definitions are in how they are applied. With a look, you also specify the "process space" - the colorspace in which the transform is applied.

Example configuration

Step 1: Setup a Look

A look is a top-level OCIO configuration object. Conceptually, it's a named transform which gets applied in a specific color space. All of the changes below to the .ocio configs can be done manually by editing the text, or using the Python API.

Example look definition in a OCIO config:

```
looks:
- !<Look>
  name: di
  process_space: rclg16
  transform: !<FileTransform> {src: look_di.cc, interpolation: linear}
```

Example Python API call:

```
look = OCIO.Look(name='di', processSpace='rclg16')
t = OCIO.FileTransform('look_di.cc', interpolation=OCIO.INTERP_LINEAR)
look.setTransform(t)
config.addLook(look)
```

The src file can be any LUT type that OCIO supports (in this case, it's a file that contains the `<ColorCorrection>` element from a CDL file.) You could also specify a .3dl, etc.

Once you define a look in your configuration, you'll see that the `OCIOLookTransform` node in Nuke will provide the named option. In this example, the 'DI' look conceptually represents a look that will be applied in DI. Other look names we often used are 'onset', 'editorial', etc. The `process_space` specifies that the transform should be applied in that space. In this example, if you provide linear input to the `OCIOLookTransform` node, the pixels will be converted to `rclg16` before applying the `look_di.cc` file-transform.

Step 2: Update the Display to use a look.

You can specify an optional 'looks' tag in the View tag, which will apply the specified look(s). This lets application in the viewer provide options which use the looks.

Example YAML config:

```
displays:
  DLP:
    - !<View> {name: Raw, colorspace: nc10}
    - !<View> {name: Log, colorspace: rclg10}
    - !<View> {name: Film, colorspace: p3dci16}
    - !<View> {name: Film DI, colorspace: p3dci16, looks: di}
  sRGB:
    - !<View> {name: Raw, colorspace: nc10}
    - !<View> {name: Log, colorspace: rclg10}
    - !<View> {name: Film, colorspace: srgb10}
    - !<View> {name: Film DI, colorspace: srgb10, looks: di}
```

Example Python API call:

```
for name,colorspace,look in [ ['Raw','nc10',''], ['Log','rclg10',''],
        ['Film','p3dci16',''], ['Film DI','p3dci16','di'] ]:
    config.addDisplay('DLP',name,colorspace,look)

for name,colorspace,look in [ ['Raw','nc10',''], ['Log','rclg10',''],
        ['Film','srgb10',''], ['Film DI','srgb10','di'] ]:
    config.addDisplay('sRGB',name,colorspace,look)
```

Option for advanced users: The looks tag is actually a comma-delimited list supporting +/- modifiers. So if you you wanted to specify a View that undoes DI, and then adds Onset, you could do "-di,+onset".

Step 3: Get per-shot looks supported.

In the top example, look_di.cc, being a relative path location, will check each location in the config's search_path. The first file that's found will be used.

So if your config contains:

```
search_path: luts
```

... then only the 'luts' subdir relative to the OCIO config will be checked.

However if you specify:

```
search_path: /shots/show/$SHOT/cc/data:luts
```

...the directory '/shots/show/$SHOT/cc/data/' will be evaluated first, and only if not found will the 'luts' directory be checked.

env-vars, absolute, and relative paths can be used both in the config's `search_path`, as well as the View's src specification.

Example:

```
- !<Look>
  name: di
  process_space: rclg16
  transform: !<FileTransform> {src: looks/$SHOT_di/current/look_$SHOT_di.cc,␣
↪interpolation: linear}
```

Note that if the per-shot lut is not found, you can control whether a fallback LUT succeeds based on if it's in the main location. You can also use this for multiple levels (show, shot, etc).

Advanced option: If some shots use .cc files, and some use 3d-luts currently there's no simple way to handle this. What we'd recommend as a work around is to label all of your files with the same extension (such as .cc), and then rely on OCIO's resilience to misnamed lut files to just load them anyways. Caveat: this only works in 1.0.1+ (commit sha-1: 6da3411ced)

Advanced option: In the Nuke OCIO nodes, you often want to preview looks 'across shots' (often for reference, same-as, etc). You can override the env-vars in each node, using the 'Context' menu. For example, if you know that $SHOT is being used, in the context key1 you should specify 'SHOT', and the in value1 specify the shot to use (such as dev.lookdev). You can also use expressions, to say parse a shot name out of `[metadata "input/filename"]`

Advanced option: If you are writing your own OCIO integration code, `getProcessor` will fail if the per-shot lut is not found, and you may want to distinguish this error from other OCIO errors. For this reason, we provide OCIO::ExceptionMissingFile, which can be explicitly caught (this can then handled using `OCIO::DisplayTransform::setLooksOverride()`). I'd expect image flipbook applications to use this approach.

5.4.10 Contexts Example

Warning: This section is from OCIO v1 and has not been updated yet.

OCIO's allows different LUT's or grades to be applied based on the current context.

These contexts are usually based on environment variables, but also allows on-the-fly context switching in applications that operate on multiple shots (such as playback tools)

Typically these would be used as part of the display transform, to apply shot-specific looks (such as a CDL color correction, or a 1D grade LUT)

A contrived example

The simplest way to explain this feature is with examples. Say we have two shots, `ab-123` and `sf-432`, and each shot requires a different LUT to view. The current shot name is stored in the environment variable SHOT.

In the OCIO config, you can use this SHOT environment variable to construct the LUT's path/filename. This path can be absolute (e.g `/example/path/${SHOT}.spi1d`), or relative to any directory on the OCIO search path, which includes the resource path (e.g `${SHOT}.spi1d`)

This is a simplified example, to demonstrate the context feature. Typically this "contextual LUT" would be used in conjunction with other LUT's (e.g before a scene-linear to log transform, followed by a 3D film emulation LUT), this will be covered in context_per_shot_grade

So, we have our empty OCIO config in `~/showcfg`, and our two LUTs in `~/showcfg/luts` which are named `af-123.spi1d` and `sf-432.spi1d`:

```
~/showcfg/
    config.ocio
    luts/
        af-123.spi1d
        sf-432.spi1d
```

In the config, we first specify the config version, and the resource path (usually this is relative to the directory containing `config.ocio`, although can be an absolute path):

```
ocio_profile_version: 1
resource_path: luts
```

Next, we define a colorspace that transforms from the show reference space to the display colorspace:

```
colorspaces:
  - !<ColorSpace>
    name: srgb8
    family: srgb
    bitdepth: 8ui
    from_reference: !<FileTransform> {src: ${SHOT}.spi1d}
```

Then add a display alias for this transform:

```
displays:
  - !<Display> {device: sRGB, name: "Shot LUT", colorspace: srgb8}
```

Finally, we point the OCIO env-variable to the config, set the SHOT env-variable to the shot being worked on, and launch Nuke (or any other OCIO-enabled application):

```
export OCIO=~/showcfg/config.ocio
export SHOT=af-123
nuke
```

In Nuke, we create an OCIODisplay node, select our "sRGB" device with the "Shot LUT" transform, and this will apply the `af-123.spi1d` LUT.

Per-shot grades

Similarly to LUTs, we use a `.cc` file (an XML file containing a single ASC CDL <ColorCorrection>), or a `.ccc` file (an XML file containing multiple ASC CDL color corrections, each with a unique ID)

The `.cc` file is applied identically to a regular LUT files, using a `FileTransform`. For example, if we have `af-123.cc` in the `luts/` directory:

```
<ColorCorrection id="mygrade">
        <SOPNode>
             <Slope>2 1 1</Slope>
             <Offset>0 0 0</Offset>
             <Power>1 1 1</Power>
        </SOPNode>
        <SatNode>
             <Saturation>1</Saturation>
        </SatNode>
  </ColorCorrection>
```

We wish to apply this grade on the scene-linear image, then transform into log and apply a 3D print emulation LUT. Since this requires multiple transforms, instead of using a single `FileTransform`, we use a `GroupTransform` (which is is just a collection of other transforms):

```
colorspaces:
  - !<ColorSpace>
    name: lnh
    family: ln
    bitdepth: 16f:
    isdata: false
```

(continues on next page)

(continued from previous page)

```
  - !<ColorSpace>
    name: lg10
    family: lg
    bitdepth: 10ui
    isdata: false
    to_reference: !<FileTransform> {src: lg10.spi1d, interpolation: nearest}

  - !<ColorSpace>
    name: srgb8
    family: srgb
    bitdepth: 8ui
    isdata: false
    from_reference: !<GroupTransform>
      children:
        - !<FileTransform> {src: ${SHOT}.cc}
        - !<ColorSpaceTransform> {src: lnh, dst: lg10}
        - !<FileTransform> {src: film_emulation.spi3d, interpolation: linear}
```

A .ccc file is a collection of <ColorCorrection>'s. The only difference is when defining the `FileTransform`, you must specify the `cccdid` key, which you can also construct using the context's environment variables. This means we could create a `grades.ccc` file containing the grade for all our shots:

```
<ColorCorrectionCollection xmlns="urn:ASC:CDL:v1.2">
      <ColorCorrection id="af-123">
              <SOPNode>
                   <Slope>2 1 1</Slope>
                   <Offset>0 0 0</Offset>
                   <Power>1 1 1</Power>
              </SOPNode>
              <SatNode>
                   <Saturation>1</Saturation>
              </SatNode>
        </ColorCorrection>
        <ColorCorrection id="mygrade">
                <SOPNode>
                     <Slope>0.9 0.7 0.9</Slope>
                     <Offset>0 0 0</Offset>
                     <Power>1 1 1</Power>
                </SOPNode>
                <SatNode>
                     <Saturation>1</Saturation>
                </SatNode>
          </ColorCorrection>
</ColorCorrectionCollection>
```

And the colorspace definition to utilise this:

```
- !<ColorSpace>
  name: srgb8
  family: srgb
  bitdepth: 8ui
  isdata: false
```

(continues on next page)

(continued from previous page)

```
  from_reference: !<GroupTransform>
    children:
      - !<FileTransform> {src: grades.ccc, cccid: ${SHOT}}
      - !<ColorSpaceTransform> {src: lnh, dst: lg10}
      - !<FileTransform> {src: film_emulation.spi3d, interpolation: linear}
```

A complete example

Warning: This is incomplete, the lnh_graded space is likely wrong

The context feature can be used to accommodate complex grading pipelines. In this example, we have a "neutral grade" for each shot, to neutralise color casts and exposure variations, keeping plates consistent throughout a sequence.

To view a shot, we reverse this neutral grade, apply a "beauty grade", then apply the display transform (the usual lin-to-log and a film emulation LUT)

We will use the same two example shots from before, af-123 (which is in the af sequence) and sg-432 (in the sg sequence). Imagine we have many shots in each sequence, so we wish to put the grades for each sequence in a separate file.

Using the same directory structure as above, in `~/showcfg/luts` we first create two grade files, `grades_af.ccc`:

```
<ColorCorrectionCollection xmlns="urn:ASC:CDL:v1.2">
      <ColorCorrection id="af/af-123/neutral">
              <SOPNode>
                   <Slope>2 1 1</Slope>
                   <Offset>0 0 0</Offset>
                   <Power>1 1 1</Power>
              </SOPNode>
              <SatNode>
                   <Saturation>1</Saturation>
              </SatNode>
        </ColorCorrection>

      <ColorCorrection id="af/af-123/beauty">
              <SOPNode>
                   <Slope>1.5 1.2 0.9</Slope>
                   <Offset>0 0 0</Offset>
                   <Power>1 1 1</Power>
              </SOPNode>
              <SatNode>
                   <Saturation>0.8</Saturation>
              </SatNode>
        </ColorCorrection>

        <!-- More ColorCorrection's... -->
</ColorCorrectionCollection>
```

And `grades_sg.ccc`:

```
<ColorCorrectionCollection xmlns="urn:ASC:CDL:v1.2">
        <ColorCorrection id="sg/sg-432/neutral">
                <SOPNode>
                     <Slope>0.9 0.7 0.9</Slope>
                     <Offset>0 0 0</Offset>
                     <Power>1 1 1</Power>
                </SOPNode>
                <SatNode>
                     <Saturation>1</Saturation>
                </SatNode>
          </ColorCorrection>

        <ColorCorrection id="sg/sg-432/beauty">
                <SOPNode>
                     <Slope>1.1 0.9 0.8</Slope>
                     <Offset>0 0 0</Offset>
                     <Power>1.2 0.9 1.5</Power>
                </SOPNode>
                <SatNode>
                     <Saturation>1</Saturation>
                </SatNode>
          </ColorCorrection>

          <!-- More ColorCorrection's.. -->
</ColorCorrectionCollection>
```

Next, we create the `config.ocio` file, containing a colorspace to define several colorspaces:

- `lnh`, the scene-linear, 16-bit half-float space in which compositing will happen
- `lg10`, the 10-bit log space in which material will be received (e.g in .dpx format)
- `srgb8`, the display colorspace, for viewing the neutrally graded footage on an sRGB display
- `srgb8graded`, another display colorspace, for viewing the final "beauty grade"

```
ocio_profile_version: 1

# The directory relative to the location of this config
resource_path: "luts"

roles:
  scene_linear: lnh
  compositing_log: lgf

displays:
  # Reference to display transform, without reversing the working grade
  - !<Display> {device: sRGB, name: Film1D, colorspace: srgb8}

  # Reference to display, reversing the working grade, and applying
  # the beauty grade
  - !<Display> {device: sRGB, name: Film1DGraded, colorspace: srgb8graded}

colorspaces:
```

(continues on next page)

(continued from previous page)

```
  # The source space, containing a log to scene-linear LUT
  - !<ColorSpace>
    name: lg10
    family: lg
    bitdepth: 10ui
    isdata: false
    to_reference: !<FileTransform> {src: lg10.spi1d, interpolation: nearest}

  # Our scene-linear space (reference space)
  - !<ColorSpace>
    name: lnh
    family: ln
    bitdepth: 16f
    isdata: false

  # Neutrally graded scene-linear
  - !<ColorSpace>
    name: lnh_graded
    family: ln
    bitdepth: 16f
    isdata: false
    to_reference: !<FileTransform> {src: "grades_${SEQ}.ccc", cccid: "${SEQ}/${SHOT}/
↪neutral"}


  # The display colorspace - how to get from scene-linear to sRGB
  - !<ColorSpace>
    name: srgb8
    family: srgb
    bitdepth: 8ui
    isdata: false
    from_reference: !<GroupTransform>
      children:
        - !<ColorSpaceTransform> {src: lnh, dst: lg10}
        - !<FileTransform> {src: lg_to_srgb.spi3d, interpolation: linear}

  # Display color, with neutral grade reversed, and beauty grade applied
  - !<ColorSpace>
    name: srgb8graded
    family: srgb
    bitdepth: 8ui
    isdata: false
    from_reference: !<GroupTransform>
      children:
        - !<FileTransform> {src: "grades_${SEQ}.ccc", cccid: "${SEQ}/${SHOT}/neutral",␣
↪direction: inverse}
        - !<FileTransform> {src: "grades_${SEQ}.ccc", cccid: "${SEQ}/${SHOT}/beauty",␣
↪direction: forward}
        - !<ColorSpaceTransform> {src: lnh, dst: srgb8}
```

5.4.11 How to Configure ColorSpace Allocation

The allocation / allocation vars were utilized using during GPU 3dlut / shader text generation in OCIO v1. However in OCIO v2, the GPU renderer does not need to bake color transforms and the allocation vars are not utilized unless an application requests the legacy GPU shader. However, the allocation vars are still utilized by the ociobakelut command-line utility.

If, in the course of baking, a 3D lut is required, the "allocation / allocation vars" direct how OCIO should sample the colorspace, with the intent being to maintain maximum fidelity and minimize clamping.

Currently support allocations / variables:

ALLOCATION_UNIFORM::
2 vars: [min, max]

ALLOCATION_LG2::
2 vars: [lg2min, lg2max] 3 vars: [lg2min, lg2max, linear_offset]

So say you have an srgb image (such as an 8-bit tif), where you know the data ranges between 0.0 - 1.0 (after converting to float). If you wanted to apply a 3d lut to this data, there is no danger in sampling that space uniformly and clamping data outside (0,1). So for this colorspace we would tag it:

```
allocation: uniform
allocationvars: [0.0, 1.0]
```

These are the defaults, so the tagging could also be skipped.

But what if you were actually first processing the data, where occasionally small undershoot and overshoot values were encountered? If you wanted OCIO to preserve this overshoot / undershoot pixel information, you would do so by modifying the allocation vars.

```
allocation: uniform
allocationvars: [-0.125, 1.125]
```

This would mean that any image data originally within [-0.125, 1.125] will be preserved during GPU processing. (Protip: Data outside this range *may* actually be preserved in some circumstances - such as if a 3d lut is not needed - but it's not required to be preserved).

So why not leave this at huge values (such as [-1000.0, 1000.0]) all the time? Well, there's a cost to supporting this larger dynamic range, and that cost is reduced precision within the 3D luts sample space. So in general you're best served by using sensible allocations (the smallest you can get away with, but no smaller).

Now in the case of high-dynamic range color spaces (such as float linear), a uniform sampling is not sufficient because the max value we need to preserve is so high.

Say you were using a 32x32x32 3d lookup table (a common size). Middle gray is at 0.18, and specular values are very much above that. Say the max value we wanted to preserve in our coding space is 256.0, each 3d lut lattice coordinates would represent 8.0 units of linear light! That means the vast majority of the perceptually significant portions of the space wouldn't be sampled at all!

unform allocation from 0-256: 0 8.0 16.0 … 240.0 256.0

So another allocation is defined, lg2

```
- !<ColorSpace>
  name: linear
  description: |
      Scene-linear, high dynamic range. Used for rendering and compositing.
  allocation: lg2
  allocationvars: [-8, 8]
```

In this case, we're saying that the appropriate ways to sample the 3d lut are logarithmically, from 2^-8 stops to 2^8 stops.

Sample locations: 2^-8: 0.0039 2^-7: 0.0078 2^-6: 0.0156 … 2^0: 1.0 … 2^6: 64.0 2^7: 128.0 2^8: 256.0

Which gives us a much better perceptual sampling of the space.

The one downside of this approach is that it can't represent 0.0, which is why we optionally allow a 3d allocation var, a black point offset. If you need to preserve 0.0 values, and you have a high dynamic range space, you can specify a small offset.

Example:

```
allocation: lg2
allocationvars: [-8, 8, 0.00390625]
```

The [-15.0, 6.0] values in spi-vfx come from the fact that all of the linearizations provided in that profile span the region from 2^-15 stops, to 2^6 stops. One could probably change that black point to a higher number (such as -8), but if you raised it too much you would start seeing black values be clipped. Conversely, on the high end one could raise it a bit but if you raised it too far the precision would suffer around gray, and if you lowered it further you'd start to see highlight clipping.

5.4.12 Developing with OCIO

5.4.13 Usage Examples

Some examples of using the OCIO API, via both C++ and the Python bindings.

For further examples, see the `src/apps/` directory in the git repository

Applying a basic ColorSpace transform, using the CPU

This describes what code is used to convert from a specified source `ColorSpace` to a specified destination `ColorSpace`. If you are using the OCIO Nuke plugins, the OCIOColorSpace node performs these steps internally.

1. **Get the Config.** This represents the entirety of the current color "universe". It can either be initialized by your app at startup or created explicitly. In common usage, you can just query `GetCurrentConfig()`, which will auto initialize on first use using the *`OCIO`* environment variable.
2. **Get Processor from the Config.** A processor corresponds to a 'baked' color transformation. You specify two arguments when querying a processor: the colorspace_section you are coming from, and the colorspace_section you are going to. cfgcolorspaces_section ColorSpaces can be either explicitly named strings (defined by the current configuration) or can be cfgroles_section (essentially colorspace_section aliases) which are consistent across configurations. Constructing a `Processor` object is likely a blocking operation (thread-wise) so care should be taken to do this as infrequently as is sensible. Once per render 'setup' would be appropriate, once per scanline would be inappropriate.
3. **Get a CPU or GPU Processor.** Building the processor assembles the operators needed to perform the requested transformation, however it is not ready to process pixels. The next step is to create a CPU or GPU processor. This optimizes and finalizes the operators to produce something that may be executed efficiently on the target platform.
4. **Convert your image, using the Processor.** Once you have a CPU or GPU processor, you can apply the color transformation using the "apply" function. In usage_applybasic_cpp, you may apply the processing in-place, by first wrapping your image in an `ImageDesc` class. This approach is intended to be used in high performance applications, and can be used on multiple threads (per scanline, per tile, etc). In usage_applybasic_python you

call “applyRGB” / “applyRGBA” on your sequence of pixels. Note that in both languages, it is far more efficient to call “apply” on batches of pixels at a time.

C++

```
#include <OpenColorIO/OpenColorIO.h>
namespace OCIO = OCIO_NAMESPACE;

try
{
   OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();
   OCIO::ConstProcessorRcPtr processor = config->getProcessor(OCIO::ROLE_COMPOSITING_LOG,
                                                              OCIO::ROLE_SCENE_LINEAR);

   OCIO::ConstCPUProcessorRcPtr cpu = processor->getDefaultCPUProcessor();

   // Apply the color transform to an existing RGBA image
   OCIO::PackedImageDesc img(imageData, width, height, 4);
   cpu->apply(img);
}
catch(OCIO::Exception & exception)
{
   std::cerr << "OpenColorIO Error: " << exception.what() << std::endl;
}
```

Python

```
import PyOpenColorIO as OCIO

try:
   config = OCIO.GetCurrentConfig()
   processor = config.getProcessor(OCIO.ROLE_COMPOSITING_LOG,
                                   OCIO.ROLE_SCENE_LINEAR)
   cpu = processor.getDefaultCPUProcessor()

   # Apply the color transform to the existing RGBA pixel data
   img = [1, 0, 0, 0]
   img = cpu.applyRGBA(img)
except Exception as e:
   print("OpenColorIO Error: ", e)
```

Displaying an image, using the CPU using getProcessor

1. **Get the Config.** See usage_applybasic for details.
2. **Get the desired display and view.** If the user has specified a display and view, use them. Otherwise `Config::getDefaultDisplay()` and `Config::getDefaultView()` may be used to get the defaults. These may be provided directly in a call to getProcessor rather than the source and destination color space.
3. **Get the processor and CPU processor from the Config.** See usage_applybasic for details.
4. **Convert your image, using the processor.** See usage_applybasic for details.

C++

```
#include <OpenColorIO/OpenColorIO.h>
namespace OCIO = OCIO_NAMESPACE;

try
{
   OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

   const char * display = config->getDefaultDisplay();
   const char * view = config->getDefaultView(display);

   OCIO::ConstProcessorRcPtr processor = config->getProcessor(OCIO::ROLE_SCENE_LINEAR,
                                                              display, view,
                                                              OCIO::TRANSFORM_DIR_
↪FORWARD);
   OCIO::ConstCPUProcessorRcPtr cpu = processor->getDefaultCPUProcessor();

   OCIO::PackedImageDesc img(imageData, width, height, 4);
   cpu->apply(img);
}
catch(OCIO::Exception & exception)
{
   std::cerr << "OpenColorIO Error: " << exception.what() << std::endl;
}
```

Python

```
import PyOpenColorIO as OCIO

try:
   config = OCIO.GetCurrentConfig()

   display = config.getDefaultDisplay()
   view = config.getDefaultView(display)

   processor = config.getProcessor(OCIO.ROLE_SCENE_LINEAR, display, view, OCIO.TRANSFORM_
↪DIR_FORWARD)
   cpu = processor.getDefaultCPUProcessor()
```

(continues on next page)

(continued from previous page)

```
    img = [1, 0, 0]
    cpu.applyRGB(img)
except Exception as e:
    print("OpenColorIO Error: ", e)
```

Displaying an image, using the CPU using DisplayViewTransform

C++

```
// Step 1: Get the config
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

// Step 2: Lookup the display ColorSpace
const char * display = config->getDefaultDisplay();
const char * view = config->getDefaultView(display);

// Step 3: Create a DisplayViewTransform, and set the input and display ColorSpaces
// (This example assumes the input is a role. Adapt as needed.)

OCIO::DisplayTransformRcPtr transform = OCIO::DisplayViewTransform::Create();
transform->setSrc( OCIO::ROLE_SCENE_LINEAR );
transform->setDisplay( display );
transform->setView( view );

// Step 4: Create the processor
OCIO::ConstProcessorRcPtr processor = config->getProcessor(transform);
OCIO::ConstCPUProcessorRcPtr cpu = processor->getDefaultCPUProcessor();

// Step 5: Apply the color transform to an existing RGBA pixel
OCIO::PackedImageDesc img(imageData, width, height, 4);
cpu->apply(img);
```

Python

```
import PyOpenColorIO as OCIO

# Step 1: Get the config
config = OCIO.GetCurrentConfig()

# Step 2: Lookup the display ColorSpace
display = config.getDefaultDisplay()
view = config.getDefaultView(display)

# Step 3: Create a DisplayViewTransform, and set the input, display, and view
# (This example assumes the input is a role. Adapt as needed.)

transform = OCIO.DisplayViewTransform()
transform.setSrc(OCIO.ROLE_SCENE_LINEAR)
```

(continues on next page)

(continued from previous page)

```
transform.setDisplay(display)
transform.setView(view)

# Step 4: Create the processor
processor = config.getProcessor(transform)
cpu = processor.getDefaultCPUProcessor()

# Step 5: Apply the color transform to an existing RGB pixel
imageData = [1, 0, 0]
print(cpu.applyRGB(imageData))
```

Displaying an image, using the CPU (Full Viewport Pipeline)

This alternative version allows for a more complex viewing pipeline, allowing for all of the controls typically added to real-world viewport interfaces. For example, options are allowed to control which channels (red, green, blue, alpha, luma) are visible, as well as allowing for optional diagnostic adjustments (such as an exposure offset in scene linear).

1. **Get the Config**. In this example, use one of the built-in configs.
2. **Get the default display for this config and the display's default view.**
3. **Create a new DisplayViewTransform.** This transform has the basic conversion from the reference space to the display but without the extras such as the channel swizzling and exposure control.
4. **Set up any diagnostic or creative look adjustments.** If the user wants to specify a channel swizzle, a scene-linear exposure offset, an artistic look, this is the place to add it. See ociodisplay for an example. Note that although we provide recommendations for display, any transforms are allowed to be added into any of the slots. So if for your app you want to add 3 transforms into a particular slot (chained together), you are free to wrap them in a GroupTransform and set it accordingly!
5. **Create a new LegacyViewingPipeline.** This transform will embody the full viewing pipeline you wish to control and will add all of the specified adjustments in the appropriate place in the pipeline, including performing any necessary color space conversions. For example, the LinearCC happens in the scene_linear role of the config and the colorTimingCC happens in the color_timing role color space.
6. **Get the Processor from the LegacyViewingPipeline.** A CPUProcessor is then created from that to process pixels on the CPU.
7. **Convert your image, using the CPUProcessor.**

Python

```
import PyOpenColorIO as ocio

# Set up some example input variables to simulate a diagnostic
# adjustment a user might make using viewport controls to analyze
# different parts of the image tone scale.
exposure_val = 1.2              # +1.2 stops exposure adjustment
gamma_val = 0.8                 # adjust diagnostic gamma to 0.8

# Step 1: Use one of the built-in configs.
config = ocio.Config.CreateFromBuiltinConfig("studio-config-latest")
```

(continues on next page)

(continued from previous page)

```
# Step 2: Get the default display and view.
display = config.getDefaultDisplay()
view = config.getDefaultView(display)

# Step 3: Create a DisplayViewTransform to convert from the scene-linear
# role to the selected display & view.
display_view_tr = ocio.DisplayViewTransform(
    src=ocio.ROLE_SCENE_LINEAR,
    display=display,
    view=view
)

# Step 4: Set up any diagnostic or creative look adjustments.

# Create an ExposureContrastTransform to apply an exposure adjustment
# in the scene-linear input space. By setting the dynamic property to true,
# that allows for interactive adjustment without rebuilding the processor.
exposure_tr = ocio.ExposureContrastTransform(
    exposure=exposure_val,
    dynamicExposure=True
)

# Add a Channel view 'swizzle'.
channelHot = (1, 1, 1, 1)   # show rgb
# channelHot = (1, 0, 0, 0) # show red
# channelHot = (0, 0, 0, 1) # show alpha
# channelHot = (1, 1, 1, 0) # show luma
channel_view_tr = ocio.MatrixTransform.View(
    channelHot=channelHot,
    lumaCoef=config.getDefaultLumaCoefs()
)

# Add a second ExposureContrastTransform, this one applying an gamma
# adjustment in the output display space (useful for checking shadow
# detail).
gamma_tr = ocio.ExposureContrastTransform(
    gamma=gamma_val,
    pivot=1.0,
    dynamicGamma=True
)

# Step 5: Create a LegacyViewingPipeline which builds a processing pipeline
# by adding the various diagnostic controls around the DisplayViewTransform.
viewing_pipeline = ocio.LegacyViewingPipeline()
viewing_pipeline.setLinearCC(exposure_tr)
viewing_pipeline.setChannelView(channel_view_tr)
viewing_pipeline.setDisplayViewTransform(display_view_tr)
viewing_pipeline.setDisplayCC(gamma_tr)

# Step 6: Create a Processor and CPUProcessor from the pipeline.
proc = viewing_pipeline.getProcessor(config)
# Use the default optimization level to create a CPU Processor.
```

(continues on next page)

(continued from previous page)

```
cpu_proc = proc.getDefaultCPUProcessor()

# Step 7: Evaluate an image pixel value.
image_pixel = [0.5, 0.4, 0.3]   # a test value to process
rgb = cpu_proc.applyRGB(image_pixel)
print(rgb)
```

Displaying an image, using the GPU

Applying OpenColorIO's color processing using the GPU is very customizable and an example helper class is provided for use with OpenGL.

1. **Get the Processor and GPU Processor.** This portion of the pipeline is identical to the CPU approach. Just get the processor as you normally would have, see above for details.
2. **Create a GpuShaderDesc.**
3. **Extract the GPU Shader.**
4. **Configure the GPU State.**
5. **Draw your image.**

C++

This example is available as a working app in the OCIO source: `src/apps/ociodisplay`.

```
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

const char * display = config->getDefaultDisplay();
const char * view = config->getDefaultView(display);
const char * look = config->getDisplayViewLooks(display, view);

OCIO::DisplayViewTransformRcPtr transform = OCIO::DisplayViewTransform::Create();
transform->setSrc(OCIO::ROLE_SCENE_LINEAR);
transform->setDisplay(display);
transform->setView(view);

OCIO::LegacyViewingPipeline vpt;
vpt.setDisplayViewTransform(transform);
vpt.setLooksOverrideEnabled(true);
vpt.setLooksOverride(look);

OCIO::ConstProcessorRcPtr processor = vpt.getProcessor(config, config->
↪getCurrentContext());
OCIO::ConstGPUProcessorRcPtr gpu = processor->getDefaultGPUProcessor();

OCIO::GpuShaderDescRcPtr shaderDesc = OCIO::GpuShaderDesc::CreateShaderDesc();
shaderDesc->setLanguage(OCIO::GPU_LANGUAGE_GLSL_1_3);
shaderDesc->setFunctionName("OCIODisplay");
shaderDesc->setResourcePrefix("ocio_");
```

(continues on next page)

(continued from previous page)

```
gpu->extractGpuShaderInfo(shaderDesc);

OCIO::OglAppRcPtr oglApp = std::make_shared<OCIO::ScreenApp>("ociodisplay", 512, 512);
oglApp->initImage(imgWidth, imgHeight, OCIO::OglApp::COMPONENTS_RGBA, &img[0]);
oglApp->setShader(shaderDesc);

oglApp->redisplay();
```

Python

A full example of a GPUProcessor can be found in the OCIO source: `src/apps/pyociodisplay`.

```
import PyOpenColorIO as OCIO

# Step 1: Get the config
config = OCIO.GetCurrentConfig()

# Step 2: Lookup the display ColorSpace
display = config.getDefaultDisplay()
view = config.getDefaultView(display)

# Step 3: Create a DisplayViewTransform, and set the input, display, and view
# (This example assumes the input is a role. Adapt as needed.)
transform = OCIO.DisplayViewTransform()
transform.setSrc(OCIO.ROLE_SCENE_LINEAR)
transform.setDisplay(display)
transform.setView(view)

# Step 4: Create the processor
processor = config.getProcessor(transform)
gpu = processor.getDefaultGPUProcessor()

# Step 5: Extract the GLSL ShaderText to be used in an OpenGL app.
shader_desc = OCIO.GpuShaderDesc.CreateShaderDesc(language=OCIO.GPU_LANGUAGE_GLSL_4_0)
gpu.extractGpuShaderInfo(shader_desc)
source = shader_desc.getShaderText()
print(source)
```

5.4.14 Application Helpers

OCIO v2 includes a collection of helper code intended to ease the development of applications. This is located in src/OpenColorIO/apphelpers.

There are helpers for tasks such as:

- Building application color space menus that are able to filter by category and that are able to display hierarchical menus.
- Building a viewing pipeline for an application viewport with dynamic exposure adjustment.
- Writing a color picker that handles scene-linear color encodings well.
- Adding new color spaces or new views to a config file.

More documentation and usage examples are planned. Please stay tuned ...

5.4.15 Contributing

5.4.16 Getting started

Checking Out The Codebase

The main code repository is available on Github: http://github.com/AcademySoftwareFoundation/OpenColorIO

For those unfamiliar with git, the wonderful part about it is that even though only a limited number people have write access to the main repository, anyone is free to create, and even check in, changes to their own local git repository. Your local changes will not automatically be pushed back to the main repository, so everyone feel free to informally play around with the codebase. Also - unlike svn - when you download the git repository you have a full copy of the project's history (including revision history, logs, etc), so the majority of code changes you will make, including commits, do not require network server access.

The first step is to install git on your system. For those new to the world of git, GitHub has an excellent tutorial stepping you through the process, available at: http://help.github.com/

To check out a read-only version of the repository (no GitHub signup required):

```
git clone git://github.com/AcademySoftwareFoundation/OpenColorIO.git ocio
```

For write-access, you must first register for a GitHub account (free). Then, you must create a local fork of the OpenColorIO repository by visiting http://github.com/AcademySoftwareFoundation/OpenColorIO and clicking the "Fork" icon. If you get hung up on this, further instructions on this process are available at http://help.github.com/forking/

To check out a read-write version of the repository (GitHub acct required):

```
git clone git@github.com:$USER/OpenColorIO.git ocio

Initialized empty Git repository in /mcp/ocio/.git/
remote: Counting objects: 2220, done.
remote: Compressing objects: 100% (952/952), done.
remote: Total 2220 (delta 1434), reused 1851 (delta 1168)
Receiving objects: 100% (2220/2220), 2.89 MiB | 2.29 MiB/s, done.
Resolving deltas: 100% (1434/1434), done.
```

Both read + read/write users should then add the AcademySoftwareFoundation/OpenColorIO main branch as a remote. This will allow you to more easily fetch updates as they become available:

```
cd ocio
git remote add upstream git://github.com/AcademySoftwareFoundation/OpenColorIO.git
```

Optionally, you may then add any additional users who have individual working forks (just as you've done). This will allow you to track, view, and potentially merge intermediate changes before they're been pushed into the main trunk. (For really bleeding edge folks). For example, to add Jeremy Selan's working fork:

```
git remote add js git://github.com/jeremyselan/OpenColorIO.git
```

You should then do a git fetch, and git merge (detailed below) to download the remote branches you've just added.

Reference Build Environment

To aid new developers to the project and provide a baseline standard, OCIO provides a reference build environment through Docker. Docker essentially is a container that consists of both a Linux distro and the dependencies needed to run a client application. This is typically used for deploying apps and services to servers, but we are using it to provide an isolated development environment to build and test OCIO with. With this environment you are guaranteed to be able to compile OCIO and run its non-GUI command line applications.

For more information on Docker, start here: https://docs.docker.com/engine/docker-overview/

In order to run the Docker environment you will have to build it from the Dockerfile provided in the repo directory:

```
OpenColorIO/shared/docker
```

Run this command in order to build the Docker image (aprox. 20min):

```
docker build . -t ocio:centos7_gcc48 -f dockerfile_centos7_gcc48
```

You can then mount the current OCIO directory and compile using the Docker image with:

```
docker run --volume $PWD/../../:/src/ociosrc -t ocio:centos7_gcc48 bash -c 'mkdir /build␣
↪&& cd /build && cmake /src/ociosrc && make -j2`
```

See also *ASWF Docker* for ASWF-managed docker images for building and testing OpenColorIO.

Merging changes

After you fork and clone OCIO, the upstream repository will change continually. Your fork will not receive those changes until you manually pull and push the commits, and in the case of in-progress feature branches, you'll need to merge those changes or rebase your commits on top of them regularly to stay up to date. To update your fork from upstream run the:

```
git checkout main
git pull upstream main && git push origin main
```

The first command makes sure you have the main branch checked out, and the second combines an upstream pull (getting all the new commits to bring your local clone up to date) and an origin push (updating your fork's remote main). Following these commands the main branch will be identical between the OpenColorIO repository and your fork.

To merge these changes into a feature branch (making the branch able to merge with the upstream main), run:

```
git checkout myFeature
git merge main
```

git will report any merge conflicts encountered and allow you to resolve them locally prior to committing the merge.

Alternatively you can rebase the changes into your branch, which will replay your branch commits on top of the latest main branch commits. A rebase should only be used if you are the only contributor to a branch. Since rebasing alters the branch's commit history, a force push is required to push the changes to the remote repository, which can be problematic for others contributing to the same branch. To rebase, run:

```
git checkout myFeature
git rebase main
```

Follow the interactive instructions that git provides during the rebase to resolve merge conflicts at each replayed commit. Update your remote branch following a successful rebase with:

```
git push origin myFeature --force
```

There are various reasons why you might prefer a merge or a rebase. This article from Atlassian provides a great basis for understanding both options along with their benefits and trade-offs.

5.4.17 Repository structure

The OpenColorIO repository has a relatively straight-forward structure, and a simple branching and merging strategy.

All development work is done directly on the main branch. This represents the bleeding-edge of the project and any contributions should be done on top of it.

After sufficient work is done on the main branch and OCIO leadership determines that a release is due, we will bump the relevant internal versioning and tag a commit with the corresponding version number, e.g. v2.0.1. Each Minor version also has its own "Release Branch", e.g. RB-1.1. This marks a branch of code dedicated to that Major.Minor version, which allows upstream bug fixes to be cherry-picked to a given version while still allowing the main branch to continue forward onto higher versions. This basic repository structure keeps maintenance low, while remaining simple to understand.

Root

The root repository directory contains two groups of files:

1. Configuration files used by Git and other cloud services that interface with the OpenColorIO GitHub account.
2. Important *.md documentation on the OpenColorIO project's license, governance, and operational policies.

CMake configuration must be initiated from this directory, which is the .git root when cloning the OpenColorIO repository.

.github/workflows

This directory contains the project GitHub Actions configuration, which defines continuous integration (CI) workflows for all supported platforms. Each .yml file defines one GHA workflow, which can contain one or more jobs and the repository events that trigger the workflow.

ASWF

This subdirectory contains important ASWF governance documents like the OpenColorIO charter, CLAs and DCO reference.

The `meetings` subdirectory contains historical meeting minutes for TSC (Technical Steering Committee) and working group meetings. More recent TSC and working group meeting notes are stored on the ASWF Wiki.

docs

This directory contains the OpenColorIO documentation source and build configuration.

See documentation-guidelines for information on contributing to OCIO documentation.

ext

Houses modified external dependencies which are permissible to store within the OpenColorIO repository. Permissible dependencies have compatible FOSS licenses which have documented approval by the ASWF governing board.

include

This directory contains the OpenColorIO public C++ headers, which define the publicly accessible interface (API) for OCIO. This is the best ground truth reference available for developers wanting to explore an OpenColorIO implementation in their application.

share

This directory contains supplementary resources for building and using OpenColorIO. This includes scripts used by OCIO CI, CMake, and other build system and example components. Dockerfiles are provided to assist with container-based OpenColorIO builds.

The `cmake` subdirectory contains all CMake modules and macros used by the OCIO build system, and is the best reference for troubleshooting CMake package finding issues.

src

This directory contains the OpenColorIO source code and is broken into the following subdirectories:

- **OpenColorIO**: Core library source, including all code related to transforms, ops, and file format IO.
- **apps**: Source for each OCIO command-line tool or "app" (e.g. `ociobakelut`). These tools also provide excellent example usage of integrating the OCIO library in a C++ application.
- **apputils**: Supporting utility code shared by the OpenColorIO apps.
- **bindings**: OpenColorIO language bindings for Python and Java (note that the OCIO Java bindings are not actively maintained. Contributions to update this work is welcome and appreciated).
- **libutils**: Supplementary and example helper libraries for integrating OpenColorIO into applications. Utilities for using OCIO with OpenGL and with OpenImageIO/OpenEXR are included.
- **utils**: Common header-only utilities shared by the core OpenColorIO library and the other utility sources.

tests

This directory contains the OpenColorIO test suite, both for C++ and all supported bindings. Each core *.cpp source file has a corresponding *_tests.cpp file with all of its tests. Separate tests suites for CPU and GPU functionality are provided since a GPU may not always be available. The `tests/data/files` subdirectory contains a wide array of LUT files and other raw data for testing file transforms.

vendor

This directory contains source for plugins to integrate OpenColorIO into third-party software and DCCs (e.g. Photoshop). Software vendors are encouraged to contribute their OCIO integrations back to the project at this location if practical (e.g. plugin source).

5.4.18 Architectural notes

Coming soon...

5.4.19 Coding style guide

There are two main rules when contributing to OpenColorIO:

1. When making changes, conform to the style and conventions of the surrounding code.
2. Strive for clarity, even if that means occasionally breaking the guidelines. Use your head and ask for advice if your common sense seems to disagree with the conventions.

File conventions

C++ implementation should be named `*.cpp`. Headers should be named `*.h`.

Copyright notices

All source files should begin with the standard copyright and license statement:

```
// SPDX-License-Identifier: BSD-3-Clause
// Copyright Contributors to the OpenColorIO Project.
```

The comment syntax of the statement can be adapted to the current file's language. For example, a Python file would instead use:

```
# SPDX-License-Identifier: BSD-3-Clause
# Copyright Contributors to the OpenColorIO Project.
```

Non-source files must also include a copyright and license statement, but with the follow Creative Commons license used instead of `BSD-3-Clause` (using RST syntax in this example):

```
..
  SPDX-License-Identifier: CC-BY-4.0
  Copyright Contributors to the OpenColorIO Project.
```

Line length

Each line of text in your code should be at most 100 characters long. Generally the only exceptions are for comments with example commands or URLs - to make copy and paste easier. The other exception is for those rare cases where letting a line be longer (and wrapping on an 100-character window) is actually a better and clearer alternative than trying to split it into two lines. Sometimes this happens, but it's rare.

DO NOT alter somebody elses code to re-wrap lines (or change whitespace) just because you found something that violates the rules. Let the group/author/leader know, and resist the temptation to change it yourself.

Formatting

- Indent 4 spaces at a time, and use actual spaces, not tabs. This is particularly critical for Python code. The only exception currently allowed is within Makefiles, where tab characters are sometimes required.
- Opening brace should go on the line following the condition or loop.
- The contents of namespaces are not indented.
- Function names should be on the same line as their return values.
- Function calls should NOT have a space between the function name and the opening parenthesis. A single space should be added after each required comma.

Here is a short code fragment that shows these concepts in action:

```
namespace OCIO_NAMESPACE
{

int MungeMyNumbers(int a, int b)
{
    int x = a + b;

    if (a == 0 || b==0)
    {
        x += 1;
        x += 2;
    }
    else
    {
        for (int i=0; i<16; ++i)
        {
            x += a * i;
        }
    }

    return x;
}

} // namespace OCIO_NAMESPACE
```

Misc. rules

- Avoid macros when possible.
- Anonymous namespaces are preferred when sensible.
- Variable names should be camelCase, as well as longAndExplicit.
- Avoid long function implementations. Break up work into small, manageable chunks.
- Use `TODO:` comments for code that is temporary, a short-term solution, or good-enough but not perfect. This is vastly preferred to leaving subtle corner cases undocumented.
- Always initialize variables on construction.
- Do not leave dead code paths around (this is what revision history is for).
- Includes should always be ordered as follows: C library, C++ library, other libraries' .h, OCIO public headers, OCIO private headers. Includes within a category should be alphabetized.
- The C++ `using` directive is not allowed.
- Static / global variables should be avoided at all costs.
- Use `const` whenever it makes sense to do so.
- The use of Boost is not allowed.
- Default function arguments are not allowed.
- Class members should start with `m_` and use camelCase.
- Public static functions should be PascalCase.

Bottom line

When in doubt, look elsewhere in the code base for examples of similar structures and try to format your code in the same manner.

Portions of this document have been blatantly lifted from OpenImageIO, and Google.

5.4.20 Unit tests

All functionality in OpenColorIO must be covered by an automated test. Tests should be implemented in a separate but clearly associated source file under the tests subdirectory (e.g. tests for `src/OpenColorIO/Context.cpp` are located in `tests/cpu/Context_tests.cpp`). This test suite is collectively expected to validate the behavior of every part of OCIO:

- Any new functionality should be accompanied by a test that validates its behavior.
- Any change to existing functionality should have tests added if they don't already exist.

The test should should be run, via `ctest`, before submitting a pull request. Pull requests will not be merged until tests are present, running and passing as part of the OpenColorIO CI system.

For verbose test output (listing each test and result status), run:

```
ctest -V
```

5.4.21 Test framework

C++

The OCIO C++ test framework is based on the OpenImageIO test framework.

The macros defined in UnitTest.h are used to define tests and assert expected behavior:

- `OCIO_ADD_TEST`: Define a test and add it to the named test `group`. Added tests will execute whenever `ctest` is run. Test groups and names should be clearly named for logical test execution and identification of failures that occur.
- `OCIO_CHECK_*`: Macros for assertion of expected test results and behavior. Checks are provided for result comparison, exception handling, and various degrees of numeric precision validation.

C++ unit test source files are organized into the following subfolders:

- **apphelpers**: Tests for utilities in `src/libutils/apphelpers`.
- **cpu**: Main OCIO library tests, including functionality which utilizes the CPU renderer.
- **gpu**: GPU-specific tests, which must be run on supported graphics hardware. GPU tests are gated behind a dedicated CMake option `OCIO_BUILD_GPU_TESTS` (which is `ON` by default).
- **utils**: Tests for utilities in `src/utils`.

Python

The OCIO Python test suite uses the Python-bundled unittest framework. Each bound class has a dedicated `TestCase` class with a `test_*` method per specific test.

Test cases must be imported into OpenColorIOTestSuite.py and added to the `suite` method to be included in `ctest` execution.

Java

The OCIO Java test suite uses the JUnit framework. Each bound class extends the `TestCase` class with individual assertions begin defined in a single `test_interface` method.

Note: The OCIO Java bindings have not been actively maintained so the test suite is incompatible with the current OpenColorIO API. Contributions to restore support for Java in OCIO are welcome. See GitHub issue #950 for more information.

5.4.22 Issues

Please visit http://github.com/AcademySoftwareFoundation/OpenColorIO/issues for an up to date listing of bugs, feature requests, etc.

See the "Good first issue" label for good issues for new contributors to OpenColorIO.

5.4.23 Submitting an issue

To submit a feature request or defect/bug, visit the OCIO GitHub issues page and click the "New Issue" button.

For bug reports, include detailed information on your environment, including OCIO version or branch, OS version, compiler version, and complete steps to reproduce the issue. Small self-contained code snippets should be included to help reproduce code-related defects.

If the issue you are reporting is related to a security vulnerability, please follow the reporting procedures outlined in the OpenColorIO security policy.

5.4.24 Submitting Changes

Code Review

Ask early, and ask often!

All new contributors are highly encouraged to post development ideas, questions, or any other thoughts to *Slack* or the *Mailing Lists* before starting to code. This greatly improves the process and quality of the resulting library. Code reviews (particularly for non-trivial changes) are typically far simpler if the reviewers are aware of a development task beforehand. (And, who knows? Maybe they will have implementation suggestions as well!)

Development and Pull Requests

This will outline the general mechanics of working with git and GitHub to successfully contribute to the project. If any of the terms used are unfamiliar to you please do a quick search and then ask any of the contributors for assistance.

Tip: There's an excellent tutorial demonstrating how to contribute to an OSS GitHub project on Rob Allen's DevNotes. It is highly recommended to give this post a read prior to cloning the OpenColorIO repository if you intend to contribute.

- Fork the AcademySoftwareFoundation/OpenColorIO repository
- Clone your fork to your local workspace:

```
git clone https://github.com/$USER/OpenColorIO.git
```

- `cd` into the cloned directory
- Connect your cloned repo to the original upstream repository as a remote:

```
git remote add upstream https://github.com/AcademySoftwareFoundation/OpenColorIO.git
```

- You should now have two remotes:

```
git remote -v
origin https://github.com/$USER/OpenColorIO (fetch)
origin https://github.com/$USER/OpenColorIO (push)
upstream https://github.com/AcademySoftwareFoundation/OpenColorIO (fetch)
upstream https://github.com/AcademySoftwareFoundation/OpenColorIO (push)
```

- Pull the latest changes from upstream:

```
git checkout main
git pull upstream main
```

- Create a branch for your contribution:

```
git checkout -b myFeature
```

- Check if it successfully compiles and passes all unit tests
- Commit your changes. The `-s` option will sign-off your commit, which is a requirement for contributing to OpenColorIO. Every commit MUST be signed-off by someone authorized to contribute under a current ICLA or CCLA:

```
git add .
git commit -s -m 'Implement my feature'
```

- If your PR changes anything that appears in the public API documentation, you should follow the directions below in "Updating the Python docs". If this proves problematic for you, please reach out for help on Slack. One of the other developers will probably be able to make the updates for you in another PR.
- Push your changes back to origin (your fork):

```
git push -u origin myFeature
```

- Ensure that all CI tests complete on GitHub actions
- Visit your fork in a web browser on github.com
- When ready click the "New pull request" button, make sure it can merge, and add appropriate comments and notes. GitHub will also provide a convenient "Compare & pull request" button for recent pushes to streamline this step.
- Wait for code review and comments from the community.
- At various points in the review process other pull requests will have been merged into OpenColorIO main, at which point merging of your PR will be blocked with the message "This branch is out-of-date with the base branch". Clicking the "Update branch" button will automatically merge the updated branch into your feature branch. This step may need to be repeated multiple times depending on current commit activity.

See CONTRIBUTING.md for more information.

5.4.25 Documentation guidelines

OpenColorIO is documented using reStructuredText, processed by Sphinx.

The documentation primarily lives in the `docs/` folder, within the main OpenColorIO repository.

The rST source for the C++ API documentation is extracted from comments in the public header files in `include/`

Installation of requirements

Scripts are available, for each platform, to install the documentation requirements.

The `install_docs_env.sh` script in the `share/ci/scripts/<Platform>` directory will install the Python-related requirements for building the documentation (Sphinx, six, testresources, recommonmark, sphinx-press-theme, sphinx-tabs, and breathe) and Doxygen.

Note: If you are on Linux using yum and don't already have Doxygen installed, you will have to uncomment the relevant line in `share/ci/scripts/linux/dnf/install_docs_env.sh`

Use GitBash (provided with Git for Windows) to execute the script on Windows.

Python 3 is required to build the documentation. If you have multiple Python installs you'll need to make sure pip and CMake find the correct version. You can manually inform CMake of which to use by adding this option to the below *cmake* command, which configures the documentation build:

```
-DPython_ROOT=<Path to Python 3 root directory>
```

You need to make sure these Python package locations are in your `PYTHONPATH` environment variable prior to configuring the build. A good way to accomplish this is with a virtual environment, which would look like:

```
$ python3 -m venv venv
$ source venv/bin/activate
$ share/ci/scripts/<platform>/install_docs_env.sh  # macos or windows
or
$ share/ci/scripts/linux/<tool>install_docs_env.sh  # for linux, tool is apt or yum
$ export PYTHONPATH=</path/to>/venv/lib/python<version>/site-packages
```

Obviously, adjust for specific paths and python versions. Also, the above assumes a `bash` environment - the commands may be slightly different for other shells.

Building the docs

The build is just like a *regular build from source*, but specify the `-D OCIO_BUILD_DOCS=ON` argument to CMake.

Then run the `make docs` target. The default HTML output will be created in `build_dir/docs/build-html/`

Note that CMake must be run before each invocation of `make` to copy the edited rST files.

Initial run:

```
$ mkdir build && cd build
```

Then after each change you wish to preview:

```
$ cmake -D OCIO_BUILD_DOCS=ON ../ && make docs
```

Tip:

The `-j` option to `make` is your friend, eg, `make -j 8` will make things go much faster. The exact number will depend on the resources of your particular machine. The `nproc` command (linux) will help you decide.

Updating the Python docs

The conf.py has a switch that detects when it is being run on RTD, and in that case will itself run Doxygen to generate the XML needed by breathe prior to building the docs, and will also facilitate a CMake configure_file-like process (via Python) to handle substitutions in headers and docs source files that CMake would usually handle, but can't in this case. One potential plus to all of this is that if someone wants to just build OCIO docs, they can technically do so by running sphinx-build in the docs directory, and nothing more. Right now that only works when the READTHEDOCS env var == True, but it could be easily exposed another way if needed.

These features required several custom Sphinx extensions tuned for our project which are located under `share/docs`.

Building the docs – Excluding the API docs

If you don't need to build the API documentation, there is a quick and dirty way to do a docs build. This approach does not need to compile the C++ code but is not ideal since it modifies files in the source directory rather than the build directory:

```
export READTHEDOCS=True
cd docs  (in the source directory)
mkdir _build
sphinx-build -b html . _build
<your web browser name> _build/index.html
```

Basics

- Try to keep the writing style consistent with surrounding docs.
- Fix all warnings output by the Sphinx build process. An example of such an warning is:

```
checking consistency... [...]/build/docs/userguide/writing_configs.rst:: WARNING:␣
↪document isn't included in any toctree
```

- Use the following hierarchy of header decorations:

```
Level 1 heading
===============

Level 2 heading
***************

Level 3 heading
+++++++++++++++

Level 4 heading
---------------
```

- To add a new page, create a new `.rst` file in the appropriate location. In that directory's `index.rst`, add the new file to the `toctree` directive.

  The new file should contain a top-level heading (decorated with `=====` underline), and an appropriate label for referencing from other pages. For example, a new file `docs/userguide/baking_luts.rst` might start like this:

```
.. _userguide-bakingluts:

Baking LUT's
============

In order to bake a LUT, ...
```

Quirks

The vuepress theme that we've migrated to has some quirks to its design. For example, it only allows two nested table of contents (TOC). So things have to be organized in a slightly different way than other sphinx projects.

The root-level *toc_redirect.rst* points to where to find the different section TOCs. The name and contents of each sections TOC is defined in that sub-directory's *_index.rst* file.

In this TOC the *:caption:* directive determines what the name of the section will be in the sidebar, and in the header of the website. The *H1* header determines the name of the page in the right/left arrows navigation bar. In a lot of cases this ends up doubling up the name on the page, but this seems unavoidable at the present time. If additional explanatory text is put in the *_index.rst* files then it shouldn't be as problematic.

The site will show all *H1* headers in the side panel by default, these then expand when selected to show all *H2* headers.

Due to the limited TOC and sidebar depth, we shouldn't be afraid of looong pages with many *H2* headings to break down the page into logical quadrants.

Emacs rST mode

Emacs' includes a mode for editing rST files. It is documented on the docutils site

One of the features it includes is readjusting the hierarchy of heading decorations (the underlines for different heading levels). To configure this to use OCIO's convention, put the following in your `.emacs.d/init.el`:

```
(setq rst-preferred-decorations
      '((?= simple 0)
        (?* simple 0)
        (?+ simple 0)
        (?- simple 0)))
```

5.4.26 OpenColorIO Doxygen Style Guide

This guide introduces a consistent style for documenting OCIO public C++ headers with Doxygen. New code should be documented with the following conventions.

Source Code Documentation Syntax

Doxygen documentation needs only to be applied to the public interface for which we want to generate OCIO API documentation files. Implementation code that does not participate in this should still be enhanced by source code comments as appropriate, but these comments are not required to follow the style outlined here.

Doxygen Tags

The following tags are generally supported:

- `brief`: Describes function or class succinctly, ideally in one sentence.
- `param`: Describes function parameters.
- `tparam`: Describes a template parameter.
- `return`: Describes return values.
- `ref`: A reference to another method/class.

- `note`: Describes a note you want a user to be attentive of.
- `warning`: Describes a warning to bring a users attention to.
- `code{<extension>}`: The start of a code block. The extension denotes the language e.g. (".py", ".cpp")
- `endcode`: The end of a code block.

Additional Doxygen tags may be supported, but will need to be tested by building the Sphinx docs. Not all tags need to be used in every docstring.

Block Comments

Where possible please try to split the tag and names from the descriptive text.

```
/**
 * \brief A brief description.
 *
 * Detailed description. More detail.
 * \see Some reference
 *
 * \param <name>
 *      Parameter description.
 * \tparam <name>
 *      Template parameter description.
 * \return
 *      Return value description.
 */
```

Example

```
/**
 * \brief Returns a compressed version of a string.
 *
 * Compresses an input string using the foobar algorithm.
 *
 * \param uncompressed
 *      The input string.
 * \return
 *      A compressed version of the input string.
 */
std::string compress(const std::string& uncompressed);
```

Single Line Comments

Single line comments should be equivalent in scope to the *brief* in the block comments above. It should be reserved for functions that don't have parameters, or members that don't require much description.

```
/// A brief description.
```

Example

```
/// A default constructor
MyClass();
```

In-Line Comments

In-line comments serve the same purpose as the single line comments, but allows for greater structural flexibility.

```
///< A brief description.
```

Example

```
enum EnumType
{
  int EVal1,     ///< enum description 1.
  int EVal2      ///< enum description 2.
};
```

Documentation Language

Remember that the docstrings serve both the C++ and Python documentation. Try to use generic language where possible, without referring to C++ or Python constructs exclusively. If you do need to describe additional details for a specific implementation, please refer to which language you are speaking about.

5.5 Configurations

5.5.1 OCIO Studio Config for ACES

The OCIO Studio Config for ACES is the successor to the widely used *ACES config* for OCIO v1.

It contains the complete set of ACES color spaces, displays, and views. In addition, it contains some extra color spaces that are not officially part of ACES but that are widely used in the VFX, animation, games, and post-production industries.

Users who need a simpler config that contains just the basics needed to use ACES color management in common DCC tools are encouraged to check out the *OCIO CG Config for ACES*.

The most recent version of the Studio config is based on ACES 2.0.

The latest version of this config may be downloaded from the Releases page of its GitHub repo.

The Studio Config leverages the high quality ACES implementation built into OCIO itself and so requires no external LUT files. In fact, even the config file is built into OCIO and users may access it from any application that uses OCIO 2.2 or higher by using one of the following strings in place of the config path:

`ocio://studio-config-v4.0.0_aces-v2.0_ocio-v2.5` (for OCIO 2.5 or higher)

`ocio://studio-config-v2.2.0_aces-v1.3_ocio-v2.4` (for OCIO 2.4 or higher)

`ocio://studio-config-v2.1.0_aces-v1.3_ocio-v2.3` (for OCIO 2.3 or higher)

`ocio://studio-config-v1.0.0_aces-v1.3_ocio-v2.1` (for OCIO 2.2 or higher)

These new configs adopt an advanced naming convention so that they can be uniquely identified:

```
studio-config-v1.0.0_aces-v1.3_ocio-v2.1.ocio
|-----|-------------|---------|---------|---|
|Type | Colorspaces | ACES | Profile | |
```

Where:

- Type: The type of the config, e.g., CG or Studio
- Colorspaces: The version for the color spaces and other config features
- ACES: The aces-dev version being used (currently supports ACES 1.3 and 2.0)
- Profile: Minimum required OCIO version

In addition, the latest version of the Studio config may be accessed simply using:

`ocio://studio-config-latest`

The OCIO Configs Working Group collected input from the community and simplified the naming scheme relative to the earlier OCIO v1 ACES configs. However, aliases have been added so that the original color space names continue to work (if there is an equivalent space in the new config).

Please note that with OCIO v2 we are trying to be more rigorous about what constitutes a "color space". For this reason, the new configs do not bake view transforms or looks into the display color spaces. Therefore, it is necessary to use a DisplayViewTransform rather than a ColorSpaceTransform if you want to bake in an ACES Output Transform. This is not only more rigorous from a color management point of view, it also helps clarify to end-users the important role of a view transform in the process. Baking in a view transform is a fundamentally different process than just converting between color space encodings, and it should be perceived as such by users.

5.5.2 OCIO CG Config for ACES

The OCIO Computer Graphics (CG) config for ACES is a simple, lightweight config intended for use in typical digital content creation (DCC) apps that need robust choices for texture and rendering spaces and a basic selection of display and view transforms.

Users who need a more extensive set of color spaces, including digital cinema camera color spaces and a wider set of displays and view should look at the *OCIO Studio Config for ACES*.

Please note that some of the color spaces (e.g. for texturing) are not officially part of the ACES specifications but are included because they are widely used in VFX, animation, and games.

The most recent version of the CG config is based on ACES 2.0.

The latest version of this config may be downloaded from the Releases page of its GitHub repo.

The CG Config leverages the high quality ACES implementation built into OCIO itself and so requires no external LUT files. In fact, even the config file is built into OCIO and users may access it from any application that uses OCIO 2.2 or higher by using one of the following strings in place of the config path:

`ocio://cg-config-v4.0.0_aces-v2.0_ocio-v2.5` (for OCIO 2.5 or higher)

`ocio://cg-config-v2.2.0_aces-v1.3_ocio-v2.4` (for OCIO 2.4 or higher)

`ocio://cg-config-v2.1.0_aces-v1.3_ocio-v2.3` (for OCIO 2.3 or higher)

`ocio://cg-config-v1.0.0_aces-v1.3_ocio-v2.1` (for OCIO 2.2 or higher)

These new configs adopt an advanced naming convention so that they can be uniquely identified:

```
studio-config-v1.0.0_aces-v1.3_ocio-v2.1.ocio
|-----|-------------|---------|---------|---|
|Type | Colorspaces | ACES | Profile | |
```

Where:

- Type: The type of the config, e.g., CG or Studio
- Colorspaces: The version for the color spaces and other config features

- ACES: The aces-dev version being used (currently supports ACES 1.3 and 2.0)
- Profile: Minimum required OCIO version

In addition, the latest version of the CG config may be accessed simply using:

`ocio://cg-config-latest`

and this is the default built-in config for the library:

`ocio://default`

The OCIO Configs Working Group collected input from the community and simplified the naming scheme relative to the earlier OCIO v1 ACES configs. However, aliases have been added so that the original color space names continue to work (if there is an equivalent space in the new config).

Please note that with OCIO v2 we are trying to be more rigorous about what constitutes a "color space". For this reason, the new configs do not bake view transforms or looks into the display color spaces. Therefore, it is necessary to use a DisplayViewTransform rather than a ColorSpaceTransform if you want to bake in an ACES Output Transform. This is not only more rigorous from a color management point of view, it also helps clarify to end-users the important role of a view transform in the process. Baking in a view transform is a fundamentally different process than just converting between color space encodings, and it should be perceived as such by users.

5.5.3 ocio-v2_demo

Note: this is not intended to be a complete production-ready config, its purpose is to introduce many of the new features in OCIO v2.

Due to the limitations of the web template, it may be easier for you to read this config in a text editor. The config file is located here.

```
ocio_profile_version: 2

#
# Note: this is not intended to be a complete production-ready config, its purpose is to
# introduce many of the new features in OCIO v2.
#

description: A config to demo the new features in OCIO v2.
name:        OCIOv2-demo_2021-02-03

#
# This config does not require any external files.
#
search_path:

#
# Defining the environment (even if empty) in the config can be a performance boost.
#
environment: {}

#
# The family separator is the character used to seprate colorspace family attributes␣
↪into
# tokens or levels for hierarchical menus.  (The "/" character is also the default.)
#
```

(continues on next page)

(continued from previous page)

```
family_separator: /

#
# Roles
#
roles:
  #
  # Please define the two interchange roles.  This will allow OCIO to convert pixels␣
↪that are in
  # a color space from another config (that also implements these roles) into a color␣
↪space in
  # this config.
  #
  aces_interchange: ACES2065-1
  cie_xyz_d65_interchange: CIE-XYZ D65
  #
  # (As this is just a demo, not all the usual roles have been defined.)
  #
  color_timing: ACEScct
  compositing_log: ACEScct
  data: Raw
  default: sRGB
  scene_linear: ACEScg

#
# The File Rules allow you to configure default behavior for applications importing␣
↪image files.
#
file_rules:
    #
    # The first rule says to assign the colorspace Raw to files with a .tx extension.
    # The colorspace Raw, defined below, essentially means no color processing is␣
↪applied.
    #
  - !<Rule> {name: tx, colorspace: Raw, pattern: "*", extension: tx}
    #
    # The next rule assigns the colorspace "ARRI LogC" to any files that contain "LogC"␣
↪in the path.
    #
  - !<Rule> {name: LogC, colorspace: ARRI LogC, pattern: "*LogC*", extension: "*"}
    #
    # The next rule assigns the colorspace "sRGB" to any file that ends with .TIF or .
↪TIFF.
    #
  - !<Rule> {name: TIFF, colorspace: sRGB, regex: ".*\\.TIF?F$"}
    #
    # The next rule uses the OCIO v1 method of searching the path for all colorspaces in␣
↪the config.
    #
  - !<Rule> {name: ColorSpaceNamePathSearch}
    #
    # The rules are ordered, highest priority first.  OCIO takes the path to a file and␣
```

(continues on next page)

(continued from previous page)

```
↪applies
    # the rules one-by-one until there is a match.  The last rule, "Default", always␣
↪matches.
    # In this case the colorspace is assigned to the colorspace used by the "default"␣
↪role.
    #
  - !<Rule> {name: Default, colorspace: default}

#
# The Viewing Rules allow applications to determine an appropriate default view for a␣
↪given color
# space.  This becomes important for applications that need to display images in a␣
↪variety of color
# spaces.  For example, an ACES Output Transform might be the ideal view for scene-
↪linear or log
# encoded color spaces but not be appropriate for video color spaces.  In OCIO v2, there␣
↪is a new
# getView function that takes a display and a color space as arguments.  OCIO uses the␣
↪viewing rules
# (if present) to filter the list of views to those that are appropriate for the given␣
↪color space.
# The rules shown here mostly use the new "encoding" property of color spaces but it is␣
↪also possible
# for a rule to list color spaces directly.  Any views that do not use a rule are always␣
↪returned.
# The active_views list may be used to also sort the filtered views (or do further␣
↪filtering).
#
viewing_rules:
  #
  # The first rule means the view will only be used with colorspaces that have an␣
↪encoding set
  # to "scene-linear".
  #
  - !<Rule> {name: scene-linear, encodings: scene-linear}
  #
  # The next rule is similar but is targeted at colorspaces that have an encoding of
↪"data".
  #
  - !<Rule> {name: data, encodings: data}
  #
  # The next rule targets colorspaces that have an encoding of either "log" or "scene-
↪linear".
  #
  - !<Rule> {name: log-or-linear, encodings: [ log, scene-linear ]}
  #
  # The next rule targets colorspaces that have an encoding of either "sdr-video" or
↪"hdr-video".
  #
  - !<Rule> {name: video, encodings: [ sdr-video, hdr-video ]}
  #
  # The next rule is not used below but is an example of how a rule could target␣
```

(continues on next page)

(continued from previous page)

```
↪specific colorspaces.
  #
  - !<Rule> {name: film-log, colorspaces: Log film scan (ADX10)}


#
# The Shared Views allow you to define a view that will be reused for multiple displays.
# This can make the config easier to read and maintain.
#
shared_views:
    #
    # The first two shared views, "Log" and "Raw" have no actual dependence on the␣
↪display, so they
    # simply specify the colorspace that will be used to apply the view.
    #
    # Note that if the application implements support for viewing rules, the Log view␣
↪will only be
    # used with scene-linear colorspaces (it is typically of no value for colorspaces␣
↪that are already
    # logarithmic or that are video or data), and the Raw view will only be used with␣
↪data colorspaces.
    # (Simply removing the rule from the Raw view would make it be offered with all␣
↪colorspaces, which
    # might be preferred in some scenarios.)
    #
  - !<View> {name: Log, colorspace: ACEScct, rule: scene-linear}
  - !<View> {name: Raw, colorspace: Raw, rule: data}
    #
    # The next shared views use a view_transform and display_colorspace rather than a␣
↪simple colorspace
    # because the view does depend on the display.  By setting display_colorspace to "
↪<USE_DISPLAY_NAME>",
    # the display_colorspace will be replaced by the display colorspace below that has␣
↪the same name as
    # the display.  For example, when used with display "sRGB", the display_colorspace␣
↪is set to "sRGB".
    #
  - !<View>
    name: ACES 1.0 SDR-video
    view_transform: ACES 1.0 SDR-video
    display_colorspace: <USE_DISPLAY_NAME>
    rule: log-or-linear
    #
    # Note that YAML syntax allows views to be split onto multiple lines to make them␣
↪easier to read.
    #
    # The ordering within the shared_views section is not significant.  It is the order␣
↪they appear
    # below that determines the menu order (although active_views may be used to re-
↪order them).
    #
  - !<View>
```

(continues on next page)

(continued from previous page)

```
    name: Un-tone-mapped
    view_transform: Un-tone-mapped
    display_colorspace: <USE_DISPLAY_NAME>
    rule: log-or-linear
    #
    # The "Un-tone-mapped" and "Video" views are functionally equivalent but the viewing␣
↪rules allow
    # different names to be used based on the colorspace being viewed.  For example,␣
↪when viewing
    # video images, it's nicer to have a view called "Video" rather than "Un-tone-mapped"␣
↪(the latter
    # really only makes sense in the context of scene-referred images).
    #
  - !<View>
    name: Video
    view_transform: Un-tone-mapped
    display_colorspace: <USE_DISPLAY_NAME>
    rule: video


#
# Displays
#
displays:
  #
  # The first displays simply reference the shared views defined above.  The ordering of␣
↪the views
  # determines the menu order (unless active_views is used to re-order them).
  #
  sRGB:
    - !<Views> [ ACES 1.0 SDR-video, Un-tone-mapped, Log, Video, Raw ]
  Rec.1886 / Rec.709 video:
    - !<Views> [ ACES 1.0 SDR-video, Un-tone-mapped, Log, Video, Raw ]
  #
  # The next display uses a (non-shared) view as well as shared views.
  #
  Rec.2100-PQ:
    - !<View>
      name: ACES 1.1 HDR-video (1000 nits)
      view_transform: ACES 1.1 HDR-video (1000 nits & Rec.2020 lim)
      display_colorspace: Rec.2100-PQ
      rule: log-or-linear
    - !<Views> [ Un-tone-mapped, Log, Video, Raw ]


#
# A Virtual Display allows an application to ask OCIO to instantiate a new display from␣
↪an ICC
# monitor profile.  The ICC monitor profile essentially becomes a new display colorspace␣
↪and a
# new display is added for users, named after the ICC profile.  The new display will␣
↪inherit the
```

(continues on next page)

(continued from previous page)

```
# views from the virtual display.
#
virtual_display:
  - !<Views> [ ACES 1.0 SDR-video, Un-tone-mapped, Log, Raw ]


#
# The default view transform is the one that will be used if ColorSpaceTransform needs␣
↪to convert
# between a colorspace and a display colorspace.  (In other words, one that uses the␣
↪scene-referred
# reference space and one that uses the display-referred reference space.)
#
default_view_transform: Un-tone-mapped

#
# This section of the config defines the View Transforms.  A View Transform is a␣
↪transform that
# converts between the scene-referred reference space and the display-referred reference␣
↪space
# (which is new in OCIO v2).  The View Transform is sometimes referred to by other names␣
↪in
# various color science documents.  For example, in ACES it is called a "rendering" and␣
↪in the
# ITU standards for HDR television it is called an Opto-Optical Transfer Function (OOTF).
↪  It
# is also commonly referred to as a "tone-map".  This is typically where the S-shape␣
↪curve that
# avoids clipped highlights is applied, along with other adjustments necessary to adopt␣
↪the
# latest color science best-practices.
#
view_transforms:
    #
    # This first view transform is simply a matrix to convert from the scene-referred␣
↪reference space,
    # which is ACES2065-1 in this config, to the display-referred reference space, which␣
↪is CIE XYZ
    # with a D65 adaptive whitepoint in this config.  Rather than put the matrix␣
↪coefficients in the
    # config, it is easier to reference one of the existing BuiltinTransforms.  A␣
↪BuiltinTransform is
    # another new OCIO v2 feature, it is essentially a well-known transform that OCIO␣
↪knows how to
    # construct at runtime.
    #
  - !<ViewTransform>
    name: Un-tone-mapped
    description: |
      Convert the scene colorimetry directly to display-referred with no tone-mapping.
      This is often described as a "linear workflow." It is intended only for diagnostic␣
↪purposes.
```

(continues on next page)

(continued from previous page)

```
    from_scene_reference: !<BuiltinTransform> {style: "UTILITY - ACES-AP0_to_CIE-XYZ-D65_
↪BFD"}
    #
    # The next view transform implements the bulk of the ACES Output Transform for SDR␣
↪video.  In ACES,
    # the transform from scene-referred colorimetry to display-referred colorimetry is␣
↪identical for
    # a set of Output Transforms (e.g., sRGB and Rec.709).  Only the final conversion␣
↪from display
    # colorimetry to display code values differs.  That last part is implemented in OCIO␣
↪as a display
    # colorspace.  The BuiltinTransform here applies the RRT and first half of the ODT. ␣
↪There are
    # Builtins available in OCIO for all of the ACES Output Transforms (though for␣
↪brevity only two
    # of them are included in this demo config).
    #
  - !<ViewTransform>
    name: ACES 1.0 SDR-video
    description: |
      ACES Output Transform for SDR displays in a video viewing environment. ACES␣
↪neutrals are at D65.
    from_scene_reference: !<BuiltinTransform> {style: "ACES-OUTPUT - ACES2065-1_to_CIE-
↪XYZ-D65 - SDR-VIDEO_1.0"}
    #
    # The next view transform is very similar, implementing the bulk of the ACES Output␣
↪Transform for
    # 1000 nit HDR video. The output transform is completed via the addition of an HDR␣
↪display colorspace
    # defined below.
    #
  - !<ViewTransform>
    name: ACES 1.1 HDR-video (1000 nits & Rec.2020 lim)
    description: |
      ACES Output Transform for 1000 nit HDR displays. ACES neutrals are at D65. Gamut␣
↪limit uses Rec.2020 primaries.
    from_scene_reference: !<BuiltinTransform> {style: "ACES-OUTPUT - ACES2065-1_to_CIE-
↪XYZ-D65 - HDR-VIDEO-1000nit-15nit-REC2020lim_1.1"}

#
# Looks
#
looks:

  - !<Look>
    name: ACES-LMT - Blue light artifact fix
    description: |
      LMT for desaturating blue hues to reduce clipping artifacts
    process_space: ACES2065-1
    transform: !<BuiltinTransform> {style: "ACES-LMT - BLUE_LIGHT_ARTIFACT_FIX"}
```

(continues on next page)

(continued from previous page)

```
#
# The inactive colorspace list is another new OCIO v2 feature.  It allows you to include␣
↪colorspaces
# in the config but hide them from user menus.  In this case, we want this colorspace in␣
↪the config
# for use in the cie_xyz_d65_interchange role above, but we don't want users to be able␣
↪to select it.
# Note that you may override the inactive list via an environment variable.
#
inactive_colorspaces: [ CIE-XYZ D65 ]

#
# The Display Colorspaces are another new OCIO v2 feature.  These are like traditional␣
↪colorspaces
# but are defined in relation to a display-referred reference space rather than a scene-
↪referred space.
# One of the benefits of this is that a ColorSpaceTransform may now convert between a␣
↪pair of display
# colorspaces (e.g., from Rec.709 to sRGB) without needing to invert a tone-map all the␣
↪way back to
# the scene-referred reference space.
#
display_colorspaces:
    #
    # The display-referred reference space for this config is CIE XYZ tristimulus values␣
↪(colorimetry)
    # with an adaptive white point of D65.  As with the scene-referred reference space,␣
↪no transforms
    # are defined for it.
    #
    # This colorspace does not need to be exposed to users so it is in the inactive_
↪colorspaces list
    # above.  Also, it does not have any categories, which also means it would not be␣
↪exposed to users
    # in applications that support categories.
    #
  - !<ColorSpace>
    name: CIE-XYZ D65
    family:
    description: |
      Display connection space, CIE XYZ with D65 adaptive white point
    encoding: display-linear
    isdata: false
    categories:
    #
    # The next display colorspace is for sRGB.  It simply applies a matrix and a gamma␣
↪to convert from
    # the desired display colorimetry (in CIE XYZ space) to display code values that may␣
↪be sent to a
    # monitor.  Note that display colorspaces use the YAML keys "from_display_reference"␣
↪and
    # "to_display_reference" to specify their transforms.
```

(continues on next page)

(continued from previous page)

```
    #
  - !<ColorSpace>
    name: sRGB
    family: Displays/SDR
    description: |
      sRGB monitor (piecewise EOTF)
    isdata: false
    categories: [ file-io, basic-3d, advanced-3d, basic-2d, advanced-2d ]
    encoding: sdr-video
    from_display_reference: !<GroupTransform>
      children:
        - !<MatrixTransform> {matrix: [ 3.240969941905, -1.537383177570, -0.498610760293,
↪ 0, -0.969243636281, 1.875967501508, 0.041555057407, 0, 0.055630079697, -0.
↪203976958889, 1.056971514243, 0, 0, 0, 0, 1 ]}
        - !<ExponentWithLinearTransform> {gamma: 2.4, offset: 0.055, direction: inverse}
        - !<RangeTransform> {min_in_value: 0., min_out_value: 0., max_in_value: 1., max_
↪out_value: 1.}
    #
    # For the purposes of the demo, the preceding and following colorspaces are␣
↪implemented
    # explicitly, but they could also have been implemented using BuiltinTransforms as:
    # - !<BuiltinTransform> {style: "DISPLAY - CIE-XYZ-D65_to_sRGB"}
    # - !<BuiltinTransform> {style: "DISPLAY - CIE-XYZ-D65_to_REC.1886-REC.709"}
    #
  - !<ColorSpace>
    name: Rec.1886 / Rec.709 video
    aliases: [ rec709 ]
    family: Displays/SDR
    description: |
      Rec.709 HD broadcast monitor with Rec.1886 EOTF (gamma 2.4)
    isdata: false
    categories: [ file-io, basic-3d, advanced-3d, basic-2d, advanced-2d ]
    encoding: sdr-video
    from_display_reference: !<GroupTransform>
      children:
        - !<MatrixTransform> {matrix: [ 3.240969941905, -1.537383177570, -0.498610760293,
↪ 0, -0.969243636281, 1.875967501508, 0.041555057407, 0, 0.055630079697, -0.
↪203976958889, 1.056971514243, 0, 0, 0, 0, 1 ]}
        - !<ExponentTransform> {value: 2.4, direction: inverse}
        - !<RangeTransform> {min_in_value: 0., min_out_value: 0., max_in_value: 1., max_
↪out_value: 1.}
    #
    # This next colorspace requires the PQ curve, which would require an external LUT␣
↪file so the
    # BuiltinTransform is used in this case.  Note that the Builtin display transforms␣
↪do not clamp,
    # so it is followed by a RangeTransform which clamps to [0,1].
    #
  - !<ColorSpace>
    name: Rec.2100-PQ
    family: Displays/HDR
    description: |
```

(continues on next page)

(continued from previous page)

```
      Rec.2100-PQ monitor with Rec.2020 primaries and ST-2084 EOTF
    isdata: false
    categories: [ file-io, advanced-3d, basic-2d, advanced-2d ]
    encoding: hdr-video
    from_display_reference: !<GroupTransform>
      children:
        - !<BuiltinTransform> {style: "DISPLAY - CIE-XYZ-D65_to_REC.2100-PQ"}
        - !<RangeTransform> {min_in_value: 0., min_out_value: 0., max_in_value: 1., max_
↪out_value: 1.}


#
# The next config section is the traditional section for colorspaces that are defined␣
↪relative to a
# scene-referred connection space.  This section may be placed either before or after␣
↪the display
# colorspaces, based on which you prefer appear first in colorspace menus.
#
colorspaces:
    #
    # Note that colorspaces may have some new attributes in OCIO v2 including categories␣
↪and encoding.
    # The categories are intended to indicate which colorspace menus in an application␣
↪should include
    # a given colorspace.  The categories may come both from the application and from␣
↪the end-user.
    # Application-defined categories such as "file-io" or "working-space" indicate which␣
↪menus in the
    # application the colorspaces will appear in.  The application categories are␣
↪intended to be
    # combined with the encodings.  So for example, an application might filter by
↪"working-space" as
    # a category and "scene-linear" as an encoding to get all the scene-linear working␣
↪space options
    # to display in a certain menu.  User-defined categories such as "basic-3d" or
↪"advanced-2d"
    # indicate what type or artist should see those colorspaces.  OCIO uses the␣
↪intersection of the
    # application categories (combined with the encoding) along with the user categories␣
↪to filter
    # the list of colorspaces.  Note that the user categories are intended to be set via␣
↪an environment
    # variable based on the type of artist using the application.  OCIO does not yet␣
↪provide an official
    # list of categories, so it will be up to application developers and config authors␣
↪to determine
    # what strings to use.  OCIO does however provide a specific list of allowed␣
↪encoding strings.
    #
  - !<ColorSpace>
    name: ACEScg
    family: ACES
```

(continues on next page)

(continued from previous page)

```
    description: |
      ACEScg working space
    isdata: false
    categories: [ file-io, working-space, basic-3d, advanced-3d, basic-2d, advanced-2d ]
    encoding: scene-linear
    to_scene_reference: !<MatrixTransform> {matrix: [ 0.695452241357, 0.140678696470, 0.
↪163869062172, 0, 0.044794563372, 0.859671118456, 0.095534318172, 0, -0.005525882558, 0.
↪004025210306, 1.001500672252, 0, 0, 0, 0, 1 ]}
    #
    # OCIO v2 introduces an "aliases" property that may be used to define synonyms for␣
↪the canonical
    # colorspace name.  This may be used to define short names that are easier to embed␣
↪in file paths
    # or to handle backwards compatibility when the name of a colorspace evolves over␣
↪time.
    #
  - !<ColorSpace>
    name: ACES2065-1
    aliases: [ aces ]
    family: ACES
    description: |
      The Academy Color Encoding System reference color space
    isdata: false
    categories: [ file-io, basic-3d, advanced-3d, basic-2d, advanced-2d ]
    encoding: scene-linear
    #
    # Note that this section of the config uses "to_scene_reference" and "from_scene_
↪reference" to
    # specify the transforms, but "to_reference" and "from_reference" are also allowed␣
↪for backwards
    # compatibility with OCIO v1.
    #
  - !<ColorSpace>
    name: scene-linear Rec.709-sRGB
    family: Linear
    description: |
      Scene-linear Rec.709 or sRGB primaries
    isdata: false
    categories: [ file-io, basic-3d, advanced-3d, advanced-2d ]
    encoding: scene-linear
    to_scene_reference: !<MatrixTransform> {matrix: [ 0.439632981919, 0.382988698152, 0.
↪177378319929, 0, 0.089776442959, 0.813439428749, 0.096784128292, 0, 0.017541170383, 0.
↪111546553302, 0.870912276314, 0, 0, 0, 0, 1 ]}
    #
    # Note that in OCIO v2 it is not necessary to include most keys if they have their␣
↪default values.
    # This also applies to properties such as the interpolation method for␣
↪FileTransforms.
    #
  - !<ColorSpace>
    name: Raw
    family: Data
```

(continues on next page)

(continued from previous page)

```
    description: |
      Raw (no color processing)
    isdata: true
    categories: [ file-io, basic-3d, advanced-3d, basic-2d, advanced-2d ]
    encoding: data
    #
    # The ACEScct transform could be implemented as a Builtin using style: "ACEScct_to_
↪ACES2065-1",
    # but the implementation here is an opportunity to demo the new LogCameraTransform,␣
↪which may
    # be used to implement most of the camera-log type transforms (and is used␣
↪internally by the
    # ACEScct BuiltinTransform implementation).
    #
  - !<ColorSpace>
    name: ACEScct
    family: ACES
    description: |
      ACES camera log working space
    isdata: false
    categories: [ file-io, working-space, advanced-3d, basic-2d, advanced-2d ]
    encoding: log
    to_scene_reference: !<GroupTransform>
      children:
        - !<LogCameraTransform>
          log_side_slope:  0.05707762557077626
          log_side_offset: 0.5547945205479452
          lin_side_slope:  1.
          lin_side_offset: 0.
          lin_side_break:  0.0078125
          base: 2
          direction: inverse
        - !<MatrixTransform> {matrix: [ 0.695452241357, 0.140678696470, 0.163869062172,␣
↪0, 0.044794563372, 0.859671118456, 0.095534318172, 0, -0.005525882558, 0.004025210306,␣
↪1.001500672252, 0, 0, 0, 0, 1 ]}
    #
    # The remaining colorspaces use the BuiltinTransforms for simplicity and readability.
    #
  - !<ColorSpace>
    name: ACEScc
    family: ACES
    description: |
      ACES log working space
    isdata: false
    categories: [ file-io, advanced-2d ]
    encoding: log
    to_scene_reference: !<BuiltinTransform> {style: "ACEScc_to_ACES2065-1"}

  - !<ColorSpace>
    name: ARRI LogC
    family: Cameras/ARRI
    description: |
```

(continues on next page)

(continued from previous page)

```
      Alexa-v3-LogC-EI800 (SUP v3 color science)
    isdata: false
    categories: [ file-io, advanced-3d, basic-2d, advanced-2d ]
    encoding: log
    to_scene_reference: !<BuiltinTransform> {style: "ARRI_ALEXA-LOGC-EI800-AWG_to_
↪ACES2065-1"}

  - !<ColorSpace>
    name: RED Log3G10 / REDWideGamutRGB
    family: Cameras/RED
    description: |
      RED Log3G10 / REDWideGamutRGB
    isdata: false
    categories: [ file-io, basic-2d, advanced-2d ]
    encoding: log
    to_scene_reference: !<BuiltinTransform> {style: "RED_LOG3G10-RWG_to_ACES2065-1"}

  - !<ColorSpace>
    name: Sony SLog3 / SGamut3
    family: Cameras/Sony
    description: |
      Sony SLog3 / SGamut3
    isdata: false
    categories: [ file-io, basic-2d, advanced-2d ]
    encoding: log
    to_scene_reference: !<BuiltinTransform> {style: "SONY_SLOG3-SGAMUT3_to_ACES2065-1"}

  - !<ColorSpace>
    name: Log film scan (ADX10)
    aliases: [ ADX10 ]
    family: ACES
    description: |
      ADX10 Academy Density Exchange (10-bit printing density)
    isdata: false
    categories: [ file-io, advanced-2d ]
    encoding: log
    to_scene_reference: !<BuiltinTransform> {style: "ADX10_to_ACES2065-1"}


#
# Sometimes it is helpful to include one or more transforms in a config that are␣
↪essentially
# stand-alone transforms that do not have a fixed relationship to a reference space or a
# process space.  An example would be a "utility curve" transform where the intent is to
# simply apply a LUT1D without any conversion to a reference space.  In these cases, a
# named_transforms section may be added to the config with one or more named transforms.
#
# Note that named transforms do not show up in color space menus by default, so the
# application developer must implement support to make them available to users.
#
# This feature may be used to emulate older methods of color management that ignored the
# RGB primaries and simply applied one-dimensional transformations.  However, config␣
```

(continues on next page)

(continued from previous page)

```
↪authors
# are encouraged to implement transforms as normal OCIO color spaces wherever possible.
#
named_transforms:

  - !<NamedTransform>
    name: Utility Curve -- Cineon Log to Lin
    description: |
      Traditional Cineon-style log-to-lin with RefWhite=695, RefBlack=95, Gamma=0.6
    transform: !<LogAffineTransform>
      log_side_slope:  0.293255131965
      log_side_offset: 0.669599217986
      lin_side_slope:  0.989202248377
      lin_side_offset: 0.010797751623
      base: 10
      direction: inverse
```

5.5.4 aces_1.0.3

This section describes the ACES 1.0.3 OpenColorIO configuration.

Note: There is a more recent version of the ACES 1.0.3 config for ACES 1.2 available here:

- https://github.com/colour-science/OpenColorIO-Configs/tree/feature/aces-1.2-config

---

Note: Please note that both of these configurations have been supplanted by the new *OCIO Studio Config for ACES* for OCIO v2.

Information about ACES

The **ACES** project home page is here:

- https://acescentral.com/

The latest documentation on the ACES transforms and specifications can be found here:

- https://acescentral.com/aces-documentation/

The ACES discussion forum is here:

- https://community.acescentral.com/

Colorspaces

Colorspaces in this configurations are grouped into the following families: ACES, ADX, Look, Output, Input, Utility, Aliases. Descriptions for the colorspaces in the different families are provided below.

For ease of use across a broader number of applications, the family name of each colorspace is pre-pended to the colorspace name when the configuration is authored. Those prefixes will be omitted in this document, but will show up when the configuration is loaded and used.

ACES Colorspaces

- ACES2065-1
- ACEScc
- ACEScct
- ACESproxy
- ACEScg

Description

Colorspaces and transforms representing the core ACES working and interchange colorspaces.

Technical information

Transforms generated based on the ACES CTL Transforms

Output Colorspaces

- sRGB
- sRGB (D60 sim.)
- Rec.709
- Rec.709 (D60 sim.)
- Rec.2020
- Rec.2020 ST2048 (1000 nits)
- DCDM (P3 gamut clip)
- DCDM
- P3-D60 ST2048 (1000 nits)
- P3-D60 ST2048 (2000 nits)
- P3-D60 ST2048 (4000 nits)
- P3-D60
- P3-DCI

Description

Colorspaces and transforms implementing the ACES Output Transforms. These colorspaces produce code values ready for display on hardware devices calibrated to the standard used to name the colorspace.

Technical information

- Transforms generated based on the ACES CTL Transforms
- All transforms produce full-range output. Host applications should be used to apply an full-to-legal scaling needed.

Input Colorspaces

There are a variety of Input Transforms covering different cameras manufacturers, gamuts, transfer functions and camera settings. See below for specifics.

Description

Colorspaces and transforms that implement the ACES Input Transforms. These colorspaces are used to convert from camera-specific formats and encodings to ACES.

Technical information

References and descriptions are provided for each group of Input Transforms below.

- **The colorspaces whose names include a transfer function and a gamut name are full implementations of ACES Input Transforms.**
  - Ex. The ARRI 'V3 LogC (EI160) - Wide Gamut' colorspace
  - Ex. The RED 'REDlogFilm - DRAGONcolor2' colorspace
  - Ex. The Canon 'Canon-Log - DCI-P3 Daylight' colorspace
- The colorspaces that start with 'Linear - ' will convert to or from a specific gamut but not apply a transfer function.
- The colorspaces that start with 'Curve - ' will apply a transfer function but not convert between gamuts.

ADX Colorspaces

- ADX10
- ADX16

Description

Colorspaces and transforms representing the ACES ADX spaces used for film scanning and printing.

Technical information

- Transforms generated based on the ACES CTL Transforms
- Alex Fry's ACES 0.7.1 OCIO config was also a valuable resource.

ARRI

- ARRI 'Working with ACES'
- ARRI Input Transforms
- Conversations with Joseph Goldstone of ARRI

Canon

- Canon ACES landing page
- Clog white paper
- C700 Drivers and Software
- C500 Drivers and Software
- **C300 Drivers and Software**
  - Choose *OSX Mountain Lion v10.8* to download the IDTs
- **C300 Mark II Drivers and Software**
  - Choose *OSX Mountain Lion v10.8* to download the IDTs, labeled "EOS C300 Mark II Input Transform Version 2.0 (for Cinema Gamut / BT.2020)"
- **C100 Drivers and Software**
  - Choose *OSX Mountain Lion v10.8* to download the IDTs

Panasonic

- Vlog/V-Gamut white paper
- Varicam Drivers and Software

RED

- Understanding REDlogFilm and REDgamma
- Conversations with Graeme Nattress of RED

Sony

- Sony Input Transforms

GoPro (Experimental)

- The quality and consistency of these transforms has not been verified.
- Conversations with David Newman of GoPro

Utility

Description

A collection of colorspaces that are used to facilitate the creation of LUTs and other basic functionality.

Technical information

- The 'Log2 xx nits Shaper' and 'Dolby PQ xx nits Shaper' spaces cover the linear range centered around 18% grey. The 48 nits spaces cover -6.5 stops (0.0028125) to +6.5 stops(16.291740). The 1000 nits spaces cover -12 stops to +10 stops. The 2000 nits spaces cover -12 stops to +11 stops. The 4000 nits spaces cover -12 stops to +12 stops.
- The LMT shaper spaces cover the linear range going from 10 stops below 18% grey (0.00017578125) to 6.5 stops above 18% grey (16.291740)
- The colorspaces starting with 'Linear - ' will convert to or from a specific gamut but not apply a transfer function.
- The colorspaces starting with 'Curve - ' will apply a transfer function but not convert between gamuts.

Look

Colorspaces

- ACES 1.0 to 0.1 emulation
- ACES 1.0 to 0.2 emulation
- ACES 1.0 to 0.7 emulation

Description

Colorspaces and transforms emulating the look of the ACES 0.1, 0.2 and 0.7 release.

- Should be applied to data in the ACES2065-1 colorspace.
- Should be used before an ACES Output Transform.

Technical information

Transforms generated based on the ACES CTL Transforms

Roles

Description

The role colorspaces are aliases to the colorspaces used for the *OCIO* 'roles' functionality.

Aliases

Description

The alias colorspaces are named with all lower-case letters and no spaces, dashes, parentheses or other characters that would not work well in a filename. They are only references, aliases for the base colorspaces with more user-friendly names. These spaces were added to enable OCIO's token-based colorspace / filename matching.

- These colorspaces should not generally be used by most artists.

Roles

The standard *OCIO* roles are defined. They role assignments are:

- **color_picking**: Output - Rec.709
- **color_timing**: ACEScc
- **compositing_log**: ADX10
- **data**: Raw

- **default**: ACES2065-1
- **matte_paint**: ACEScc
- **reference**: Raw
- **scene_linear**: ACEScg
- **texture_paint**: Raw

Additionally, a number of colorspaces that are gaining wider adoption have been added to the config. Their names and assignment are:

- **compositing_linear**: ACEScg
- **rendering**: ACEScg

Displays and Views

The default config has one Display named **ACES**, which contains the following Views / colorspaces:

- sRGB, colorspace: sRGB
- sRGB D60 sim., colorspace: sRGB (D60 sim.)
- DCDM, colorspace: DCDM
- DCDM P3 gamut clip, colorspace: DCDM (P3 gamut clip)
- P3-D60, colorspace: P3-D60
- P3-D60 PQ 1000 nits, colorspace: P3-D60 PQ (1000 nits)
- P3-D60 PQ 2000 nits, colorspace: P3-D60 PQ (2000 nits)
- P3-D60 PQ 4000 nits, colorspace: P3-D60 PQ (4000 nits)
- P3-DCI, colorspace: P3-DCI
- Rec.2020, colorspace: Rec.2020
- Rec.2020 ST2048 1000 nits, colorspace: Rec.2020 ST2048 (1000 nits)
- Rec.709, colorspace: Rec.709
- Rec.709 D60 sim., colorspace: Rec.709 (D60 sim.)
- Raw, colorspace: Raw
- Log, colorspace: ACEScc

Considerations for custom config generation:

- **The choice of a single Display and many Views may not align well with the implementation of OCIO in an application.**
  - If you would like to generate a config that contains multiple Displays, with a small number of Views for each, review the config generation script's '–createMultipleDisplays' option.
- **If a Look is added to the config, a new set of Views will be added, one for each of the Views listed above except Raw and Log, that includes the Look. The Views with Looks will be interleaved in the View list with the original Views.**
  - To add a custom Look to the config, review the config generation script's '–addACESLookLUT', '–addACESLookCDL', '–addCustomLookLUT' and '–addCustomLookCDL' options.

LUTs

The default resolution is 65x65x65 for the 3D LUTs and 4096 for the 1D LUTs.

OCIO LUTs The LUTs used internally by OCIO can be can be retrieved from the repository here.

Baked LUTs LUTs that can be used outside of OCIO are included in the 'baked' directory here.

- **The LUTs encode the ACES Output Transform for a specific colorspace input and are generally named:**
  - 'Ouput Transform name' for 'Input colorspace name'.extension
  - Ex. 'sRGB (D60 sim.) for ACEScc.icc'

The LUTs included in the 'baked' directory cover the following formats and applications:

- .3dl for Autodesk Flame
- .3dl for Autodesk Lustre
- .lut for SideFX Houdini
- .csp for Autodesk Maya
- .icc for Adobe Photoshop

Generating Configurations

Python Configurations can be generated by the following *Python* package: aces_1.0.3/python

Usage is described on the command line and in the package root __init__.py file.

Features exposed for customization by the user include:

- The resolution of 1D and 3D LUTs
- Inclusion of custom Looks
- Two modes of creating the list of OCIO Displays and Views
- Selection of shaper function: Log2 or Dolby PQ

CTL Source The configuration depends on the **ACES 1.0.3** release. The release contains a number of file renames and the new ACEScct color space and a number of minor bug fixes and small additions, but is otherwise very similar as the main **ACES 1.0.2** release.

The CTL is available here:

- https://github.com/ampas/aces-dev/tree/v1.0.3/transforms/ctl

Clone this repo using the following command:

- git clone –branch v1.0.3 https://github.com/ampas/aces-dev.git

Dependencies

The *Python* configuration generation package depends on the following libraries:

- **OpenImageIO: http://openimageio.org**
  - Detailed build instructions can be found here: OpenImageIO Build Instructions
- **OpenColorIO: http://opencolorio.org**
  - Detailed build instructions can be found here: OpenColorIO Build Instructions
- **CTL**: https://github.com/ampas/CTL

Building on macOS - Use the following commands to build these packages on macOS

- **OpenColorIO**
  - brew install -vd opencolorio –with-python
- **Update the homebrew repository of install scripts to make sure that OpenImageIO is included.**
  - brew tap homebrew/science
- **Optional Dependencies for OpenImageIO**
  - brew install -vd libRaw
  - brew install -vd OpenCV
- **OpenImageIO**
  - brew install -vd openimageio –with-python
- **CTL**
  - brew install -vd CTL
- **OpenColorIO, a second time. *ociolutimage* will build with *openimageio* installed.**
  - brew uninstall -vd opencolorio
  - brew install -vd opencolorio –with-python

Thanks

The script used to generate these transforms and the transforms themselves were the product of work and conversations with a number of people. Thanks go to:

- Steve Agland
- Joe Bogacz
- Jack Binks
- Scott Dyer
- Alex Fry
- Alex Forsythe
- Joseph Goldstone
- Stephen Hill
- Jim Houston
- Thomas Mansencal

- Robert Molholm
- Nikola Milosevic
- Will McCown
- Graeme Nattress
- David Newman
- Sam Richards
- Erik Strauss
- Doug Walker
- Kevin Wheatley

Author

The original author of this OCIO config is:

- Haarm-Pieter Duiker

5.5.5 spi-anim

This is a real OCIO color profile in use at Sony Pictures Imageworks, and is suitable for use on animated features. The concepts utilized in this profile have been successfully validated on a variety of Sony Pictures Animation features including Cloudy With A Chance Of Meatballs, Surf's Up, and Arthur Christmas.

5.5.6 spi-vfx

This is a real OCIO color profile in use at Sony Pictures Imageworks, and is suitable for use on visual effects (VFX) work. The concepts utilized in this profile have been successfully validated on a variety of Imageworks visual effects films, including Spider-Man, Alice In Wonderland, G-Force, and Green Lantern.

Conversion from film to/from scene-linear is a simple, trivially invertible 1D transform. The display transforms are complex, 3D film-print emulations.

In production, this profile is typically used before final color details are worked out. Although it sounds temporary, most of a film can be made off this configuration. Final color decisions for a film are often made long after significant work has been done. In some cases shots from a film can be finaled before the color details, such as which Digital Intermediate (DI) house will be used, are decided. Entire projects have been completed using this profile without modification.

This profile embodies two philosophies of color management familiar to those in production: *"Keep It Simple"*, and, *"Don't Be Evil"*.

The following steps outline a simplified visual effects color workflow:

- Load a plate (log film scan, digital motion picture camera, etc)
- Convert device color space to scene-linear
- Render and composite in scene-linear
- Convert from scene-linear to device color space
- Output final plate

It is absolutely critical to guarantee that process - end to end - is colorimetrically a no-op. Under no circumstances are any unintended modifications to the original image allowed.

Thus, this profile uses very simple (1D) conversions for all input and output color space conversions. All of the complexity (the 3D LUT film emulation lifting) is handled at display time, and is never baked (or unbaked) into the imagery. For visualization, this profile includes a generic Kodak Vision print emulation suitable for display on a reference sRGB monitor or a P3 Digital Cinema projector.

Caveot 1: Of course, we realize that there are many other color workflows that may be equally good (or better) than the one presented here. Please, if you have a successful alternative workflows share the details!

Caveot 2: We are not distributing the code that generates the luts from their underlying curve representations. While we hope to publish this eventually, at the current time this process relies on internal tools and we don't have the spare development cycles to port this code to something suitable for distribution.

Invertibility

Elements often need to be transferred back and forth many times between different colorspaces. Since it's impossible to know in advance how many times an image may be transferred between colorspaces it is essential for the majority of transformations to be lossless invertible transformations. By the end of the color pipeline even a 1 value difference in a 10bit transformation can become a significant issue. Invertible transformations can be taken from the source space, to linear and back with no change to the data. A higher value is placed on transformations being predictable and invertible than absolutely correct. All 1-d luts allow for forward and inverse transformations with no loss. Unless specified all channels are equally affected. The luts are 1 bit wider than stated, so lg8 actually defines 9 bits worth of entries. This allows the rounding in the inverse direction to be applied unambiguously (lossless).

Non-invertible transforms contain 3d lookups. 3D transformations can not be inverted due to gamut mapping issues. Non-invertible transformations are only used for final output media (such as QuickTimes) and for display purposes.

Film Emulation Inversion

Inverse film emulation luts aren't supported in a default configuration. Imageworks does not use a film emulation inversion lut for texture or matte paintings. In its place a film emulation preview lut, commonly as an ICC profile, is used. Although most film emulation luts are similar they do differ significantly. The DI facility creating final color is often chosen long after significant vfx work has been begun. The film luts the film will be finished on are not made until weeks, or days, before DI begins. So the 'true' lut that will be used for the finishing is not available until very late in the production, from a VFX perspective. There are many color gamut mapping issues that arise when inverting film to video lut. Using a film inversion lut at this stage would bake in a look that isn't quite right and is very difficult to fully un-bake. It is safer to work with images in a non-constrained way and apply a visualization that can be toggled on and off.

Scene Linear

lnf, lnh, ln16

Middle Gray: 0.18

The linear space is a photometrically linear space, representing high-dynamic range (HDR) pixel values in our virtual world. Middle gray is defined to be at 0.18. While the dynamic range of lnf is technically unbounded, pixel values for natural, well exposed imagery will often fall between +/- 8 stops from gray.

The scene linear colorspace is used for compositing, rendering, and is also the profile connection space. All colorspace conversions are made in relation how they transform into or out of scene linear.

The colors defined in linear are implicitly bounded by film negative sensitivities. The space is based off an OCN film scan where values refer to linear light at the focal plane. 0.18 in linear will correspond to a %18 percent grey card

captured on filmplane under the same lighting conditions where diffuse white is 1.0. Values above 1.0 in any channel would indicate a 'specular', or light emitting objects.

lnf is a full precision (32-bit) floating point colorspace. It is used for rendering and compositing.

lnh is a half precision (16-bit) floating point colorspace. It is used for rendering and compositing.

ln16 is a 16 bit integer representation of the [0,1] range of lnf. This is no longer used but is kept if a legacy image needs to be loaded, or if linear images need to be loaded into an application that does not support float. Note that storing a float lnf image using an integer ln16 representation is destructive, and throws away data.

Film Log

lg8, lg10, lg16, lgf

Middle Gray: 445 (of 1023)

The log to linear curve is based on an analysis of several commonly used Kodak acquisition stocks. It was found that Kodak 5218 is right about in the middle in terms of tone response given the input imagery we receive. The curve incorporates some toe compensation. The curve gamma closely matches 5218. The transformation does not represent any single stock. The Imageworks log conversions are not channel specific, all color channels are transformed uniformly. Compositing productivity gains have been found using the toe compensations when compared to using a straight line log to linear curve. Shoulder compensation - while technically correct - detracted from compositing quality, often creating situations where grain film noise would result in larger than desired changes in linear light.

lg8, lg10, and lg16 are similar. They are all the same log to linear transformation but are explicitly defined to be lossless at the specified bit depths. The luts use nearest neighbor interpolation to search for an exact match. Significant performance gains were found when using the proper bitdepth lut. While using the lg16 conversion on an 8 bit image will yield the same result, it is measurably slower than using the 8-bit conversion (assuming 8-bits is all that is needed). This performance gap remains even on current graphics hardware.

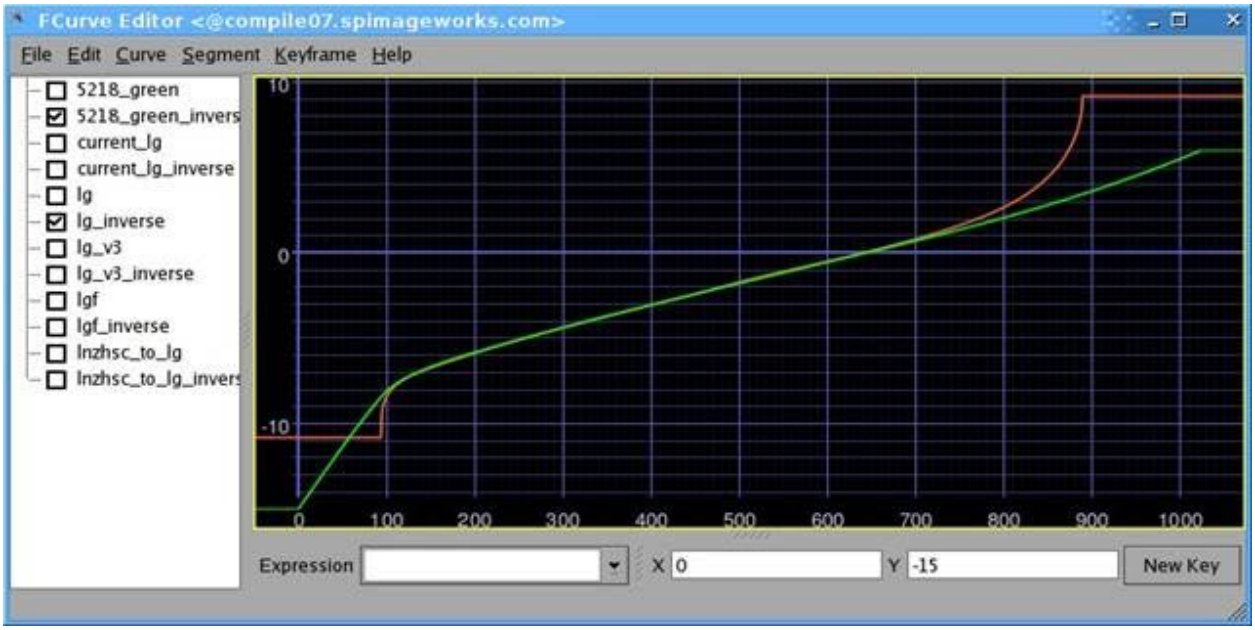


Fig. 1: lg10 to linear light. The green curve represents the lg to ln conversion. The red curve show the green channel of a Kodak 5218 exposure test. The x-axis is in 10bit input lg the output is in lg base2 linear light units. 0.0 Represents diffuse white.

lg spaces specify 21 of stops of dynamic range. 0 in lg10 space is 15 stops below diffuse white. 445 correspond to 18% grey and is ~2.47 below diffuse white. 1023 in lg10 space is 6 stops above diffuse white.

lgf is identical on the range from 0-1 to the standard lg luts. It specifies an additional range below 0.0 and above 1.0. In 10 bit the spaces is defined from -512 to 2048. Lg color timing number from either on set color correction devices or from a DI house to be applied in a way that can be fully inverted out without loss. Lgf specifies 18 stops above the lg10 max and 36 stops below the log10 min with a total dynamic range of 85 stops. The space is designed to be bigger than needed.

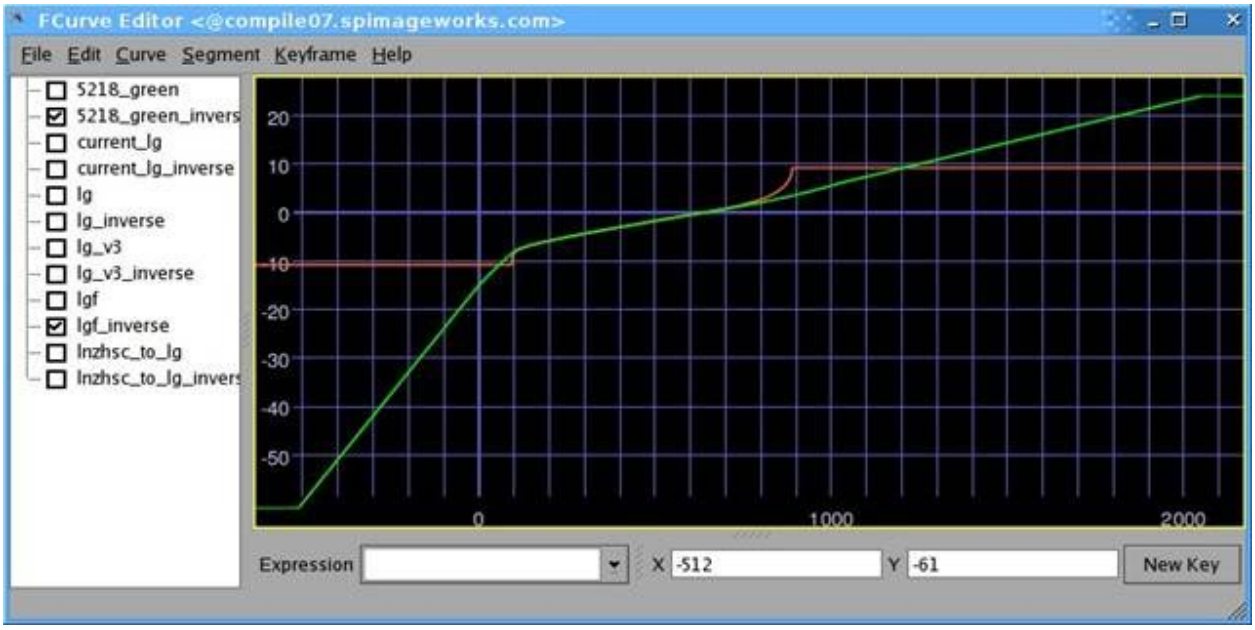


Fig. 2: lgf to linear light. The green curve represents the lg to ln conversion. The red curve show the green channel of a Kodak 5218 exposure test. The x axis is in 10bit input lg the output is in log(base2) linear light units. 0 Represents diffuse white

Panalog (Genesis)

gn8, gn10, gn16, gnf

GN is the Imageworks Panalog space. It follows the Panalog specification and additionally extrapolates so all of the values from 0 to 1023 are defined. This was necessary due to compression artifacts that could create values below the Panalog specifications. gn8,10,16 are defined with diffuse white at 681, Max white is approximately 2.6 stops above diffuse white and black is approximately 12.6 stops below diffuse white. The dynamic range is less than that of lg.

gnf is similar in purpose and function to lgf. It is identical on the range from 0-1 to the regular gn and specifies an additional range below 0.0 and above 1.0. In 10 bit numbers gnf is defined from -255 to 3125. This allows for color timing number from either on set color correction devices or from a DI house to be applied in a way that can be fully inverted. Additionally it allows for lg10 based image data to be fully represented without clipping.

gnf specifies 14.5 stops above the gn10 max and 18 of stops below the gn10 min. The entire range of gnf is 47 stops.

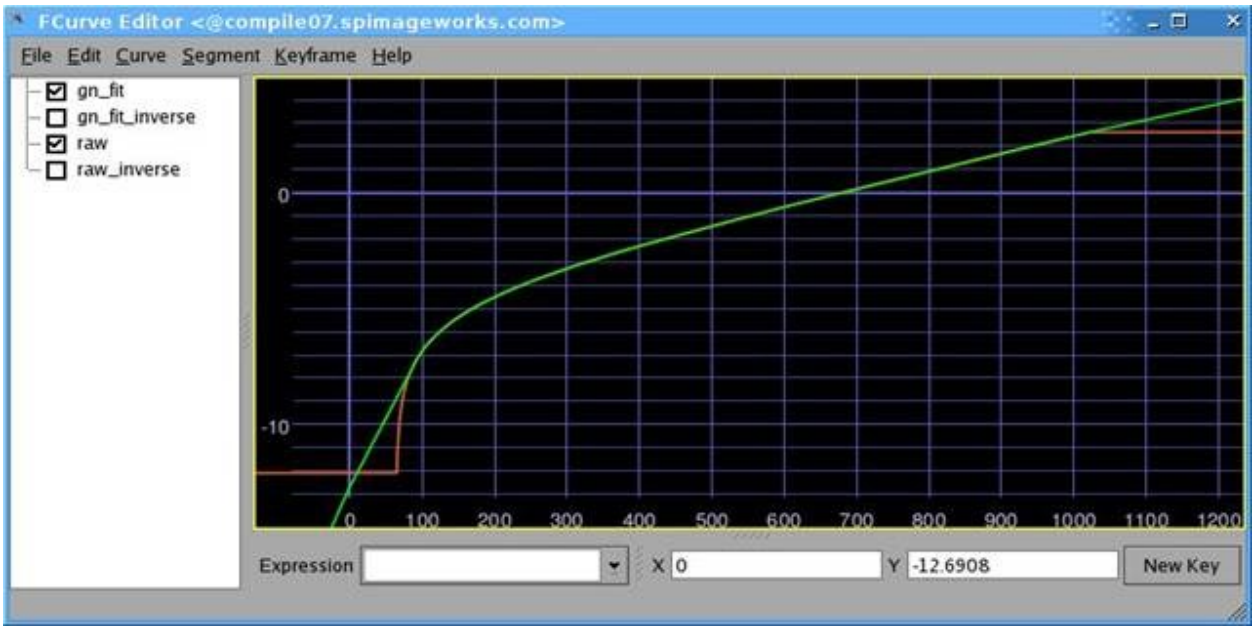


Fig. 3: gn10 to linear light. the x axis is in 10bit Panalog values. The Y axis is in linear light. The green curve is the gn curve. the red curve is the Panalog data.

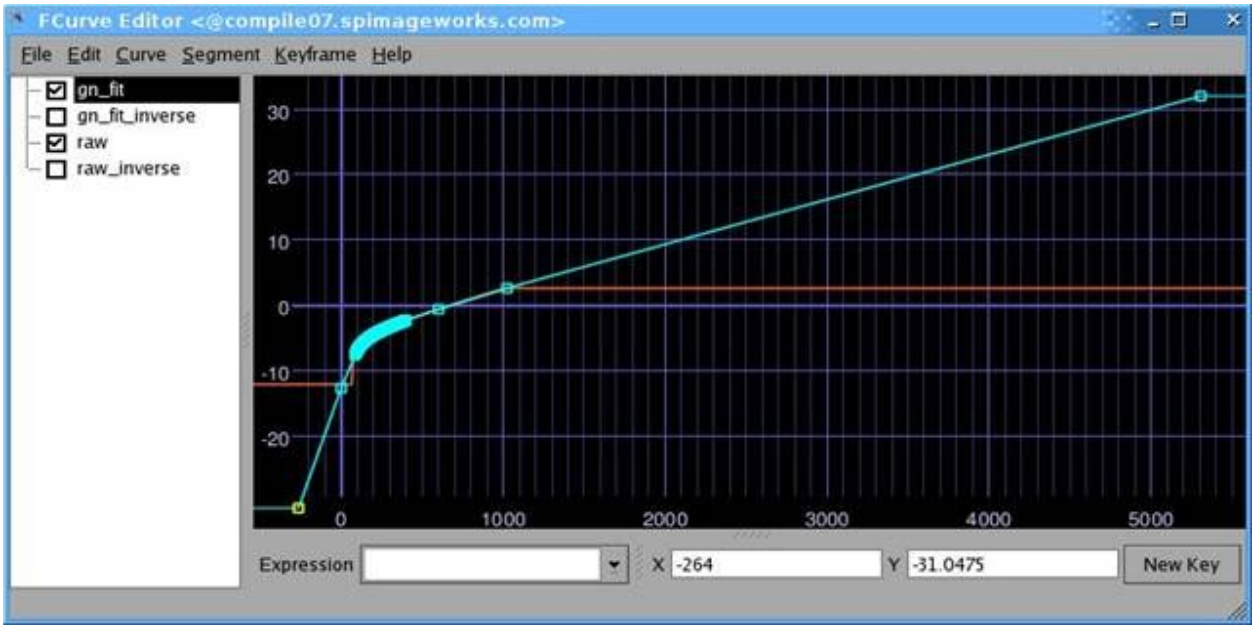


Fig. 4: gnf to linear light. the x axis is in 10bit Panalog values. The Y axis is in linear light. The green curve is the gn curve. the red curve is the Panalog data. Only a subset of the function is used to define the gnf solorspace

Reference Art

vd8, vd16, vdf, hd10

The vd spaces are mappings of linear image data into display space. The main part of the transformation is defined as a single curve that is conceptually two parts. The first is a ln to lg conversion. The second is lg to sRGB conversion. This is based off the neutral channel response of the sRGB film emulation lut used in the profile. The dynamic range of the vd colorspace is limited. It is undesirable to map the vd max to the linear max. Such a conversion results in linear values are almost never what an artist intended. The rule of thumb is that, at the high end, single value deltas in an 8 bit image should never create over a half stop of additional linear light. The vd conversion curve is limited to prevent this case.

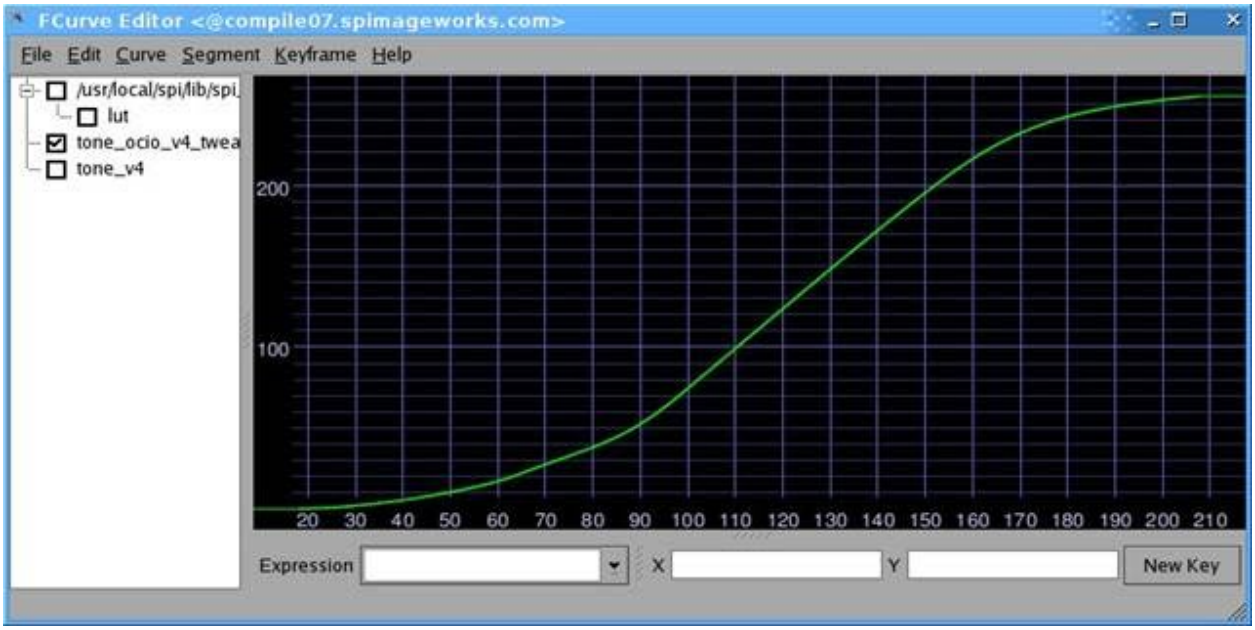


Fig. 5: The curve used to map from Lg8 to vd 8. The x-axis is in lg8 units, the y-axis is in vd8 units.

The dynamic range of the vd is limited to around 2.5 stops above diffuse white. This has two positive attributes. It allows vd to be used to directly on matte paintings. It also allows all of vd to be represented in a gn10 image. The last part of the transformation is a matrix transformation that moves the whitepoint of film to look correct when displayed with a d65 whitepoint.

The main use of this colorspace is to import RGB images with an unknown colorspace. This colorspace no longer gets much use alone; However it is an integral part of many conversions. It is also part of the matte painting and diffuse texture pipelines.

vdf works differently than the other floating spaces. It still only defines the color transformation from 0-1. This colorspace is used when we receive video space encoded exr's.

HD10 is a vd based space that is used for importing and exporting REC709 range broadcast material. This works very well for broadcast camera native material and poorly for material with a film emulation lut baked in. This transformation works well exporting film based material to tape, even though it lacks a film emulation lut. It does not give an accurate color rendering but created a pleasing image that makes clients happy.

Structurally the conversion is a matrix operation that scales the data then adds an offset to limit the range from 64-940. From there the standard vd transformation is applied.

Texture Painting

dt8, dt16

DT 8,16 - Diffuse texture colorspaces. These colorspaces are used for the conversion of textures painted in video space into a range limited linear space for use in rendering. The Color space is based on the vd transformation but limits the conversion into linear space so that no values above diffuse white can be created. This ensures that textures do not add light to a render. This is achieved by using a matrix transformation to limit the mapping of vd to the linear value of diffuse white.

Data

nc8, nc10, nc16, ncf

Non-Color (NC) spaces are used to hold any data that needs to be processed though the color pipeline unaltered. These are data spaces and can hold anything such as point clouds, normals, untagged reference art, etc. These data spaces do not get transformations applied at any point.

Display Transforms

srgb8, p3dci8, xyz16

srgb8 bakes in the film3d emulation lut. This table can be used for either QuickTime generation or output to the sRGB display. The transformation is a 3d film emulation table with gray balance compensation, so a value of 445,445,445 in lg10 space os modified to become equal RGB values in sRGB. Additionally the lut is scaled so that at least one of the color channels on maximum white uses the display max.

The transformation takes place in three parts. First the linear data is converted to the show log space. Then a film emulation table is applied. Then the grey balance and white scaling compensation are applied. This table is designed to be evaluated in a dimly lit office environment on a sRGB display.

p3dci8 is an implementation of film emulation table that has an output of DCI P3. This is only ever used for driving DLPs for display. The transformation has two parts. First the linear image data is converted to lg10 based image data then the DCI P3 film emulation lut is applied. No additional compensations are made.

xyz16 is designed for the creation of a Digital Cinema Distribution Master (DCDM). The color matches that of the P3 output (dlpqt8), but has an additional output transformation to convert to X'Y'Z'. The transformation takes the linear image data and converts it to lg, then applies the filmlook. The data is then in DCI P3 colorspace. That data is converted to display linear P3, using an inverse gamma curve. A matrix conversion is then used to transform from DCI P3' into XYZ'. The final step is to reapply the gamma 2.6 to result in XYZ16 values.

In this profile each display has three identical looks defined. The names are kept consistent between devices to minimize confusion. OCIO uses a specific tag to associate colorspaces with displays. The tags are nothing more than links to already defined colorspaces.

Film is the image displayed though a film emulation lut. This visualization is display compensated and should visually match between a sRGB display and a P3 projector. The goal is to match how the film will look in a DI. The luts in use for this profile roughly match the Sony ColorWorks environment.

Raw visualization shows the image data on the screen with no display compensation. This is used for image debugging purposes, for instance to see if potential image discontinuities are in the source data or the visualization.

Log visualization displays the image as if it were converted to the show specific log. This is transformation also has no display compensation. The common use for this us to see how well elements fit into the comp without the film emulation lut disguising any flaws. Old school compositors love it for grain matching.

sRGB Film : srgb8 sRGB Raw : nc10 sRGB Log : lg10

DCIP3 Raw : nc10 DCIP3 Log : lg10 DCIP3 Film : dlpqt8

Display Calibration

sRGB is the supported desktop display specification, calibrated to the sRGB standard and viewed in a dim office environment. As Imageworks switched from crt based display devices to LCD based devices a number of possible colorspaces were explored. It was a long decision but sRGB was chosen for a few reasons. An important one was that almost every display manufacturer can implement sRGB, reasonable well. This becomes a boon when we work needs to be done outside of our main facilities. Even a consumer display with calibration can come close to matching the sRGB standard. Since so many monitor manufacturers can hit sRGB calibration we are not tied to purchasing from a specific vendor. It becomes unnecessary to specify a specific display to with productions or external vendors. It also reduces the amount of studio specific color requirements that need to be communicated when working with other facilities. 80 cd/m^2, D65 white point, srgb gamma function (approx 2.4)

P3 was deemed especially unusable on the desktop. The full specification requires a white point of 48 cd/m^2. To adapt P3 for desktop use (in dim ambient environments), the whitepoint luminance needed to be raised. The specified 2.6 gamma is very challenging to the current display technology on very dark colors. This meant that we would have a special Imageworks video P3. A custom colorspace wouldn't make compositing better and would require a conversation, or conversion, every time video was sent out of house.

DCIP3 is a projector calibrated to DCI P3 mastering specification in a theatrical mastering environment. We use a mix of display technologies, SXRD and DLP, depending on application. Gamma 2.6. Traditional DCI calibration.

5.5.7 nuke-default

This profile corresponds to the default Nuke color configuration (currently generated from Nuke 6.1).

If you have made modifications to a nuke color configuration, and wish to re-export your own custom OCIO profile, please refer to the nuke_to_ocio utility script distributed with OpenColorIO.

The following color transforms are defined:

- linear
- sRGB
- rec709
- Cineon
- Gamma 1.8
- Gamma 2.2
- Panalog
- REDLog
- ViperLog
- REDSpace

5.6 Releases

5.6.1 OCIO 2.5 Release

Timeline

OpenColorIO 2.5 was delivered in September 2025 and is in the VFX Reference Platform for calendar year 2026.

Breaking Changes

Please be aware of the following changes when upgrading to OCIO 2.5.

For Users

- OCIOZ archive files created on Windows in prior releases should be regenerated due to an issue in a third-party library. Please see below for more details.
- For applications that use the categories attribute of color spaces to filter the color space menus shown to users, the filtering will now include color spaces that do not have any category set. (Though applications may use a new API call to override this behavior.)

For Developers

- Applications that use the OCIO GPU renderer may need to make small adjustments to their code due to changes that were required in order to support the Khronos Vulkan graphics API.
- The Python functions getActiveDisplays and getActiveViews now return iterators, please see below for details.
- Please see the section below about version updates to required third-party dependencies.

Library Version

- The 2.5.1 release is not ABI compatible with 2.5.0 for clients that use the GPU renderer API due to a change made to correct an issue with the Vulkan support introduced in 2.5.0. Because the SOVERSION of both libraries remains "2.5", applications using the GPU API that were compiled with the 2.5.0 release must be recompiled to use the 2.5.1 library. Any further releases in the 2.5.x series will be ABI-compatible with 2.5.1.

5.6.2 New Feature Guide

Config Merging (PREVIEW RELEASE)

Config files may now be merged. This was a long-standing feature request and it opens up interesting new possibilities for how configs are created, managed, and deployed. Because the workflow ramifications of this feature are far-reaching, this is being labelled a Preview Release and therefore the API and behavior may change as a result of more feedback and testing.

Five different merge strategy options are provided, along with an extensive set of parameters to customize the process. The new feature handles the complexities inherent in the very flexible OCIO config format such as:

- Handling name or alias conflicts.

- Handling differences in reference connection spaces by adding compensating color space conversions to transforms.
- Avoiding adding duplicate color spaces (even if they are named are differently). This allows applications to avoid adding color spaces that may already be present in the user's config.
- Providing a mechanism to group added color spaces under a separate menu hierarchy.

For Users

End-users may use the new `ociomergeconfigs` command-line tool to merge configs. A new OCIOM file format is provided to set the parameters that control how the merge is done. For example, you may merge a specific set of color spaces or displays and views that are needed for a given project into a general-purpose base config.

For Developers

Developers may use the new API to merge a set of color spaces that an application requires into the user's config. (This is typically done on the fly, in memory, to avoid modifying any config files on disk.) Applications may then publish their required color spaces as a config file which makes the process transparent to color scientists or pipeline experts at a facility who may use `ociomergeconfigs` to merge the application color spaces into their own configs and validate everything works as intended.

Built-in ACES 2.0 Configs

The ACES 2.0 library support was finalized in the OCIO 2.4.2 release. In the 2.5.0 release, built-in ACES 2.0 versions of the Studio and CG config are now provided. This allows use of the ACES 2.0 Output Transforms as OCIO views.

This ASWF Open Source Days presentation from ILM's Alex Fry provides a great overview of the benefits of the new configs.

In addition, the new configs make expanded use of OCIO v2 features such as File Rules and Viewing Rules and add a virtual display to support use with ICC monitor profiles.

Please take note that the display color spaces in the new configs pass through negative values rather than clamping them (by mirroring the transfer function around the origin). This provides better support for video workflows that require values below reference black to be passed through.

Finally, there are a few other color space additions including a Gamma 2.2 Rec.709 display color space and a color space for DJI Log.

For Users

The following URI strings may be provided anywhere you would normally provide a file path to a config (e.g. as the OCIO environment variable):

To use the updated *OCIO CG Config for ACES*, use this string for the config path:
ocio://cg-config-v4.0.0_aces-v2.0_ocio-v2.5

To use the updated *OCIO Studio Config for ACES*, use this string for the config path:
ocio://studio-config-v4.0.0_aces-v2.0_ocio-v2.5

This string will give you the current default config, which is the latest ACES 2 CG Config:
ocio://default

This string now points to this latest ACES 2 CG config:
ocio://cg-config-latest

This string now points to this latest ACES 2 Studio config:
ocio://studio-config-latest

Hue Curve Transform

A new `GradingHueCurveTransform` is added to allow a type of selective color correction often available in grading tools. A set of eight spline-based curve adjustments are provided:

- Hue-Hue: Map input hue to output hue (where a diagonal line is the identity).
- Hue-Sat: Adjust saturation as a function of hue (a value of 1.0 is the identity).
- Hue-Lum: Adjust luma as a function of hue (a value of 1.0 is the identity).
- Lum-Sat: Adjust saturation as a function of luma (a value of 1.0 is the identity).
- Sat-Sat: Adjust saturation as a function of saturation (a diagonal is the identity).
- Lum-Lum: Adjust luma as a function of luma, maintaining hue & sat (diagonal is identity).
- Sat-Lum: Adjust luma as a function of saturation (a value of 1.0 is the identity).
- Hue-FX : Map input hue to delta output hue (a value of 0.0 is the identity).

Unlike the typical implementation of tools of this type, this transform is invertible and has been developed to work well on video, log, and scene-linear color encodings.

The parameters may be adjusted via OCIO's dynamic parameter mechanism which allows adjustments to be made without having to recreate a Processor object. This facilitates real-time adjustment in applications.

Vulkan Support

For Developers

Support has been added for the Khronos Vulkan graphics API standard. The OCIO GPU renderer may now be used within applications that use Vulkan to get the most out of the latest GPUs.

This has required a small modification to the existing GPU API. Developers that use the GPU interface may need to modify their code when updating to OCIO 2.5. The API for the following functions have been modified: `addUniform`, `addTexture`, `add3DTexture`, and `createShaderText`. The `addToDeclareShaderCode` function has been replaced by the more specific `addToParameterDeclareShaderCode` and `addToTextureDeclareShaderCode`. In addition new functions have been added to get the descriptor set and texture binding indices. The `GPU_LANGUAGE_GLSL_VK_4_6` option is now available for the GpuLanguage enum.

New Color Space Attributes

For Config Authors

New attributes are provided to allow OCIO to better interoperate with other color standards. OCIO now supports the Color Interop ID developed by the Color Interop Forum. This allows config authors to set an ID on the color spaces in a config to unambiguously identify them as conforming to the recommendations of the ASWF Color Interop Forum. These IDs may then be used in file formats including OpenEXR and OpenUSD. (Please note that the IDs are for use in file formats but the color space's name attribute is still what should be used in a UI.)

The new attributes are:

- `interop_id`: Holds an ID string intended to be used across configs and file formats.
- `icc_profile_name`: Holds the name of an associated ICC profile.
- `amf_transform_ids`: Holds the ID strings associated with the ACES Metadata Format (AMF).

Please note that the drafts of various Color Interop Forum recommendations on this topic are currently being finalized and should be published on GitHub over the next few months.

New "edr-video" Encoding

For Config Authors

Color spaces currently have an encoding attribute that is used to give applications a general sense of how colors are distributed within the numerical encoding. There are currently two encodings that are recommended for use with video: *sdr-video* and *hdr-video*, where the latter is intended for color spaces such as Rec.2100-PQ and Rec.2100-HLG. However, there are increasingly a set of color spaces (such as "Display P3 HDR" in the built-in configs) that use what is essentially an SDR encoding that is extended so that it may be used in cases where there is a mix of both HDR and SDR content. For these color spaces, a new encoding is being introduced: *edr-video*.

Version Updates

The latest versions of all third-party dependencies have been reviewed for compatibility. In some cases, support for older versions has been dropped.

For Users

- Support for minizip-ng 3.x has been dropped. Due to a bug in previous versions of that library, any OCIOZ files created on Windows would not have been cross-platform, are not fully supported in OCIO 2.5, and should be regenerated. You may use standard Zip decompressors to expand the .ocioz files on Windows and use the *ocioarchive* command-line tool to recompress them. The new files will be cross-platform and will be compatible with both OCIO 2.5 and previous releases.

For Developers

- Support for CMake 4 has been added.
- Support for more recent versions of all dependences has been added.
- Support for C++ 11 and 14 has been dropped. OCIO now requires at least a C++ 17 compiler.
- Support for Python 3.8 has been dropped.
- Support for Expat prior to 2.6.0 has been dropped.
- Support for OpenEXR prior to 3.2.0 has been dropped.
- Support for Yaml-cpp prior to 0.8.0 has been dropped.
- Continuous Integration testing has added VFX Platform 2025 and dropped 2021.
- Miscellaneous other modifications and improvements have been made.

API Enhancements

For Developers

The previous "frozen docs" system for generating Python API documentation for ReadTheDocs has been replaced. This reduces the burden on developers when adding new API functions and simplifies the maintenance of the project.

The previous API for the active display and view lists returned one long string that developers would need to manually parse into tokens, which could be error-prone (as commas are both separators and may be used in display and view names). A new set of API calls has been provided to get and set active displays and views individually. In the case of Python, the `getActiveDisplays` and `getActiveViews` functions now return iterators, which is the only change that requires a modification to existing code.

In addition, here are some other new API functions:

- The `setDisplayTemporary` function may be used to serialize displays created using an ICC profile and a config's virtual display. This is helpful if a host application needs to serialize a config that includes those displays for use by plug-ins such as OpenFX.
- The `setTreatNoCategoryAsAny` function may be used on the ColorSpaceMenuParameters class to control the menu filtering behavior for color spaces that do not have the categories attribute set. The default is to include those color spaces when doing category filtering since this is the least surprising behavior for end-users that are editing their own configs.
- The following convenience methods have been added to the Config class for working with various types of views: `isViewShared`, `clearSharedViews`, `AreViewsEqual`, `hasView`, `hasVirtualView`, `isVirtualViewShared`, `AreVirtualViewsEqual`.
- The `setStringVar` function is now available on the Context class in Python.
- A `removeNamedTransform` function is now available.
- The previously deprecated function `getDefaultBuiltinConfigName` has now been removed. Please use `ResolveConfigPath("ocio://default")` instead.

New Fixed Function Transforms

For Config Authors

The following new styles are available for use with FixedFunctionTransforms in config files with `ocio_profile_version` set to 2.5 or higher. They implement a conversion from an RGB color space encoding to a hue, saturation, and luma space that is used by the new hue curve transform. Three conversions are provided that are optimized for scene-linear, logarithmic, and video color spaces:

- `FIXED_FUNCTION_RGB_TO_HSY_LIN`
- `FIXED_FUNCTION_RGB_TO_HSY_LOG`
- `FIXED_FUNCTION_RGB_TO_HSY_VID`

New Built-in Transforms

For Config Authors

In config files with `ocio_profile_version` set to 2.5 or higher, config authors may take advantage of the following new BuiltinTransform styles that mirror the transfer function around the origin to pass negative values rather than clamp them:

- `DISPLAY - CIE-XYZ-D65_to_REC.1886-REC.709 - MIRROR NEGS`
- `DISPLAY - CIE-XYZ-D65_to_REC.1886-REC.2020 - MIRROR NEGS`
- `DISPLAY - CIE-XYZ-D65_to_G2.2-REC.709 - MIRROR NEGS`
- `DISPLAY - CIE-XYZ-D65_to_sRGB - MIRROR NEGS`
- `DISPLAY - CIE-XYZ-D65_to_G2.6-P3-D65 - MIRROR NEGS`

5.6.3 Release Notes

For additional details, please see the GitHub release page:

OCIO 2.5.0

5.6.4 OCIO 2.4 Release

Timeline

OpenColorIO 2.4 was delivered in September 2024 and is in the VFX Reference Platform for calendar year 2025.

5.6.5 New Feature Guide

ACES 2.0 Output Transforms (PREVIEW RELEASE)

The Academy Color Encoding System has released a new major version with a completely new set of Output Transforms (which are the transforms that convert from ACES2065-1 to a display color space). These transforms are much more sophisticated than ACES 1 and include a sophisticated gamut mapping algorithm that takes advantage of recent work in color appearance modeling. The invertibility of the transforms is also improved.

ACES 2 support in OCIO 2.4.0 is labelled a "Preview Release" to indicate that this is a work-in-progress that is not yet ready for production use. Both the processing results and the API for it will likely change in OCIO 2.4.1 and future releases.

The OCIO config files for ACES 2 are still under development, but prototype configs for testing may be downloaded from this PR on the OpenColorIO-Config-ACES repo.

A preliminary technical description of the ACES 2 Output Transform algorithm may be found at this link.

Built-in Configs

An updated set of built-in CG and Studio configs is included in the 2.4 release. These include the updates from the OCIO Configs Working Group over the past year. The updates include:

- Updated texture asset color space names, following the work of the ASWF Color Interop Forum. (The previous names for all active color spaces have been preserved via an alias.)
- Making the display color spaces active (allowing for easier display-to-display conversions).

Additional updates for the Studio config include:

- Addition of the Apple Log camera color space

Given that the ACES 2 support is still a work-in-progress, the built-in configs in 2.4.0 are based on ACES 1.3. Versions for ACES 2 are under development and are planned for the OCIO 2.4.1 release.

For Users

The following URI strings may be provided anywhere you would normally provide a file path to a config (e.g. as the OCIO environment variable):

To use the updated *OCIO CG Config for ACES*, use this string for the config path:
ocio://cg-config-v2.2.0_aces-v1.3_ocio-v2.4

To use the updated *OCIO Studio Config for ACES*, use this string for the config path:
ocio://studio-config-v2.2.0_aces-v1.3_ocio-v2.4

This string will give you the current default config, which is the latest ACES CG Config:
ocio://default

This string now points to this latest CG config:
ocio://cg-config-latest

This string now points to this latest Studio config:
ocio://studio-config-latest

Tool Enhancements

For Users

- There are now Python Wheels available for Python 3.13 (Python 3.7 support will be removed soon).
- There is a `bitdepth` argument to `ocioconvert` to control the processing and writing bit-depth.
- There is a `ociocpuinfo` utility that prints info about the CPU in your computer, especially which SIMD intrinsics it supports. As with the other command-line tools, this is available as part of an OCIO install when building from source or via PyPI.

API Enhancements

For Developers

You may find the following additions to the OCIO API useful:

- There is now a `hasAlias` method on the ColorSpace and NamedTransform classes.
- The FileTransform class has a new static method `IsFormatExtensionSupported` that checks if a specific LUT format file extension is supported by OpenColorIO.

New Fixed Function Transforms

For Config Authors

The following new styles are available for use with the FixedFunctionTransform:

- `FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20` – This implements the part of the ACES 2.0 Output Transform that converts from ACES2065-1 to the intended linear values in the limiting primaries. (This requires further steps to get to the intended display colorimetry, typically a clamp, scale, and matrix.) It takes 9 parameters: peak_luminance (in nits), followed by the chromaticity coordinates for the limiting gamut: [red_x, red_y, green_x, green_y, blue_x, blue_y, white_x, white_y]. **This is a work-in-progress and may change or be removed in future releases.**
- `FIXED_FUNCTION_ACES_RGB_TO_JMH_20` – This implements the conversion from a linear RGB space into the JMh (lightness, colorfulness, hue) color appearance model space that is used in the ACES 2 Output Transform algorithm. It takes 8 parameters which are the chromaticity coordinates of the linear RGB color space: [red_x, red_y, green_x, green_y, blue_x, blue_y, white_x, white_y]. **This is a work-in-progress and may change or be removed in future releases.**
- `FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20` – This implements the part of the ACES 2 Output Transform algorithm that modify the tonescale and chroma of the image when mapping from the scene-referred to display-referred image state. It takes one parameter: the peak_luminance of the target display device (in nits). **This is a work-in-progress and may change or be removed in future releases.**
- `FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20` – This implements the part of the ACES 2 Output Transform algorithm that applies gamut compression to the target display gamut. It takes the same 9 parameters as `FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20`. **This is a work-in-progress and may change or be removed in future releases.**
- `FIXED_FUNCTION_LIN_TO_PQ` – This is the curve for SMPTE ST-2084 (PQ). It is scaled with 100 nits at 1.0. Negative values are mirrored around the origin. It takes no parameters. Note that this implementation is slower (especially on the CPU) than the Built-in Transform that implements the same function via a LUT1D rather than via a formula: `CURVE - LINEAR_to_ST-2084`. The only advantage of the Fixed Function is that it does not use texture memory on the GPU.
- `FIXED_FUNCTION_LIN_TO_GAMMA_LOG` – This curve contains a parametrized gamma and log segment. It takes 10 parameters and is capable of implementing both the Apple Log curve and the OETF for Hybrid Log Gamma (HLG). The parameters are: [mirrorPt, breakPt, gammaSeg_power, gammaSeg_slope, gammaSeg_off, logSeg_base, logSeg_logSlope, logSeg_logOff, logSeg_linSlope, logSeg_linOff].

  The exact formula may be found in the renderer code in ops/fixedfunction/FixedFunctionOpCPU.cpp

  For implementing the Apple Log or HLG OETF curves, it is recommended to use the Built-in Transforms `CURVE - APPLE_LOG_to_LINEAR` and `CURVE - HLG-OETF` as they are faster (especially on the CPU) since they are implemented via a LUT1D rather than a formula.

- FIXED_FUNCTION_LIN_TO_DOUBLE_LOG – This curve contains two parameterized LogAffineTransforms with an optional middle linear segment. It takes 13 parameters and is capable of implementing the Canon Log 2 and Canon Log 3 curves. The parameters are: [base, break1, break2, logSeg1_logSlope, logSeg1_logOff, logSeg1_linSlope, logSeg1_linOff logSeg2_logSlope, logSeg2_logOff, logSeg2_linSlope, logSeg2_linOff, linSeg_slope, linSeg_off]

  The exact formula may be found in the renderer code in ops/fixedfunction/FixedFunctionOpCPU.cpp

  For implementing the Canon Log 2 or 3 curves, it is recommended to use the Built-in Transforms CURVE - CANON_CLOG2_to_LINEAR and CURVE - CANON_CLOG3_to_LINEAR as they are faster (especially on the CPU) since they are implemented via a LUT1D rather than a formula.

New Built-in Transforms

For Config Authors

In config files with ocio_profile_version set to 2.4 or higher, config authors may take advantage of the following new BuiltinTransform styles:

- APPLE_LOG_to_ACES2065-1 – Converts Apple Log to ACES2065-1.
- CURVE - APPLE_LOG_to_LINEAR – Applies only the EOTF curve for Apple Log.
- CURVE - HLG-OETF – Applies the OETF curve for Hybrid Log Gamma.
- CURVE - HLG-OETF-INVERSE – Applies the inverse-OETF curve for Hybrid Log Gamma.
- DISPLAY - CIE-XYZ-D65_to_DCDM-D65 – Implements a display color space to produce a Digital Cinema Distribution Master (DCDM) with a D65 creative white point. This uses the traditional 2.6 gamma SDR transfer function.
- DISPLAY - CIE-XYZ-D65_to_ST2084-DCDM-D65 – Implements a display color space to produce an HDR Digital Cinema Distribution Master (DCDM) with a D65 creative white point. This implements the ST-2084 transfer function for HDR with an input Y value of 1.0 mapping to 100 nits.

The following implement various View Transforms used by the ACES 2 Output Transforms:

- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-REC709_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-108nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-300nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-500nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-1000nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-2000nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-4000nit-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-500nit-REC2020_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-1000nit-REC2020_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-2000nit-REC2020_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-4000nit-REC2020_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-REC709-D60-in-REC709-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-REC709-D60-in-P3-D65_2.0

- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-REC709-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - SDR-100nit-P3-D60-in-XYZ-E_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-108nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-300nit-P3-D60-in-XYZ-E_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-500nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-1000nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-2000nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-4000nit-P3-D60-in-P3-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-500nit-P3-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-1000nit-P3-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-2000nit-P3-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-4000nit-P3-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-500nit-REC2020-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-1000nit-REC2020-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-2000nit-REC2020-D60-in-REC2020-D65_2.0
- ACES-OUTPUT - ACES2065-1_to_CIE-XYZ-D65 - HDR-4000nit-REC2020-D60-in-REC2020-D65_2.0

5.6.6 Release Notes

For additional details, please see the GitHub release page:

OCIO 2.4.0

5.6.7 OCIO 2.3 Release

Timeline

OpenColorIO 2.3 was delivered in August 2023 and is in the VFX Reference Platform for calendar year 2024.

5.6.8 New Feature Guide

New config editor GUI app `ocioview`

There is a new GUI app for inspecting and editing OCIO configs called `ocioview`. As of the 2.3.0 release, `ocioview` is currently in an alpha release (please see the README in the apps/ocioview directory for details). Documentation and tutorials will be forthcoming and we are looking for contributors to help with this and other tasks.

Built-in Configs

An updated set of built-in CG and Studio configs is included in the 2.3 release. These include the updates from the OCIO Configs Working Group over the past year. The updates include:

- Addition of Apple's Display P3 as both a texture space and a display
- A `texture` category to allow filtering of color spaces typically used for textures
- More appropriate color space assignments for the `matte_paint` and `texture_paint` roles

Additional updates for the Studio config include:

- Support for CanonLog2 CinemaGamut color space (in addition to the existing CanonLog3)
- Addition of ACES Transform IDs to better support the ACES Metadata Format (AMF)

For Users

The following URI strings may be provided anywhere you would normally provide a file path to a config (e.g. as the OCIO environment variable):

To use the updated *OCIO CG Config for ACES*, use this string for the config path::
ocio://cg-config-v2.1.0_aces-v1.3_ocio-v2.3

To use the updated *OCIO Studio Config for ACES*, use this string for the config path::
ocio://studio-config-v2.1.0_aces-v1.3_ocio-v2.3

This string will give you the current default config, which is the latest ACES CG Config::
ocio://default

Because there are now several built-in configs, there is now a short-cut to referring to the latest versions:

This string will always give you the latest CG config::
ocio://cg-config-latest

This string will always give you the latest Studio config::
ocio://studio-config-latest

For Developers

As the built-in config collection evolves, special names such as `ocio://default` and `ocio://studio-config-latest` will point to newer versions of those configs. Therefore, it is recommended that application developers not save those strings and instead save the string that refers to the current version of that config. That way, it's guaranteed that there will be no change of behavior in the future. For example, as of OCIO 2.3, `ocio://default` should be saved as `ocio://cg-config-v2.1.0_aces-v1.3_ocio-v2.3`.

The method for doing this is to use the new ResolveConfigPath function. For example: `ResolveConfigPath("ocio://default")`

And if you intend to present a list of built-in configs to users, be sure to filter the list of configs returned by the `BuiltinConfigRegistry` and only show the ones where `isBuiltinConfigRecommended` is True so that you are only presenting the most recent versions of each of the built-in configs.

Improved Lut1D and Lut3D performance via AVX SIMD intrinsics

In some cases, this gives more than a 2x speed-up in CPU Renderer performance.

For Developers

The library now contains multiple Lut1D and Lut3D evaluation routines and the appropriate one is selected at runtime based on which SIMD instruction set is supported by the chip. Please let us know if you encounter any issues as you test on various flavors of x86 hardware.

Improved support for macOS ARM chips

For Developers

SSE SIMD intrinsics are now converted to the equivalent Neon intrinsics when building for the `arm64` platform.

In addition, the build process for generating macOS Universal Binaries has been improved.

OCIO command-line tools are now available via PyPI

For Users

Doing a `pip install opencolorio` now gives you the command-line tools such as `ociocheck`, `ociochecklut`, and `ocioconvert`.

Enhanced methods to find an equivalent color space between configs

In OCIO 2.2, several methods were introduced that used heuristics to identify a color space in a user's config that matches one of the known color spaces in the default built-in config. In OCIO 2.3, these methods were improved and extended as follows:

- `IdentifyBuiltinColorSpace` Searches a user config to find the name of an equivalent color space to a color space in one of the built-in configs. For example, "What is the name in this config for the `sRGB - Texture` color space from the default built-in config?"
- `IdentifyInterchangeSpace` Identifies the pair of color space names to be used as the interchange space to allow conversion between color spaces in a user config and one of the built-in configs. This is now the fundamental function that contains the heuristics for trying to identify known color spaces and is used by IdentifyBuiltinColorSpace and in the GetProcessorFrom/ToBuiltinColorSpace methods.

This improves upon and extends upon PR #1710. One piece of feedback on that PR is that it would be helpful to be able to use the heuristics to extract the names of the interchange color spaces rather than a Processor. This gives the application more flexibility in how that information is persisted and may reduce the number of times that any heuristics (that search through a config's color spaces) need to be run.

Finally, an issue was fixed in `GetProcessorFromConfigs` so that the returned Processor is a no-op if either of the color spaces in the two configs is a data space.

Note that if the official interchange role(s) are present (as is required in configs with `ocio_profile_version` 2.2 or higher), the above methods simply use those rather than relying on any heuristics.

Breaking API Change

For Developers

For developers that have customized their GPU Renderer by overriding the GpuShaderCreator class, there is a minor breaking change in the API. The addTexture and getTexture virtual methods now take a `TextureDimensions` parameter. This was necessary to give client apps full control over whether 1D or 2D textures are used for Lut1D ops. Please see PR #1762 for an example of how to update your implementation of these modified virtual methods.

API Enhancements

For Developers

You may find the following additions to the OCIO API useful:

- There is now a `clearProcessorCache` method to clear the Processor, CPUProcessor, and GPUProcessor contents from the cache present in a Config instance. This is useful if a LUT file on disk has changed since the Config instance was created.
- The `setAllowTexture1D` method on the GpuShaderCreator/GpuShaderDesc class allows your GPU Renderer implementation to always use 2D textures rather than 1D textures for Lut1D ops.
- The Config class now has `GetProcessorFromConfigs` variants that allow converting from a source color space in one config to a display/view in another config.
- The Config class now has an `isInactiveColorSpace` method and `getRoleColorSpace` method.
- The CPUProcessor now has the `isDynamic` and `hasDynamic` methods that were present on the Processor class.
- The `ResolveConfigPath` function converts one of the built-in config short-cut strings into the unambiguous full URI for the current config.
- The new `IdentifyBuiltinColorSpace` and `IdentifyInterchangeSpace` methods are described above.

New Built-in Transforms

For Config Authors

In config files with `ocio_profile_version` set to 2.3 or higher, config authors may take advantage of the following new transform:

- `DISPLAY - CIE-XYZ-D65_to_DisplayP3` To implement a display for the Apple Display P3 color space

5.6.9 Release Notes

For more detail, please see the GitHub release pages:

OCIO 2.3.0

5.6.10 OCIO 2.2 Release

Timeline

OpenColorIO 2.2 was delivered in October 2022 and is in the VFX Reference Platform for calendar year 2023.

5.6.11 New Feature Guide

Built-in Configs

The OCIO Configs Working Group has put a lot of effort into building new ACES configs that take advantage of the features of OCIO v2 and that are informed by the learnings from the previous ACES configs. The new configs are available from the project's GitHub repo: OpenColorIO-Config-ACES

To make them easily accessible, they are built in to the library itself and may be accessed directly from within applications that incorporate the OCIO 2.2 library.

For Users

Wherever you are able to provide a file path to a config, you may now provide a URI-type string to instead use one of the built-in configs. For example, you may use these strings for the OCIO environment variable.

To use the *OCIO CG Config for ACES*, use this string for the config path::
ocio://cg-config-v1.0.0_aces-v1.3_ocio-v2.1

To use the *OCIO Studio Config for ACES*, use this string for the config path::
ocio://studio-config-v1.0.0_aces-v1.3_ocio-v2.1

This string will give you the current default config, which is currently the ACES CG Config::
ocio://default

In future releases, it is expected that updated versions of these configs will be added, each with a unique name string. However, the previous strings will continue to be supported for backwards compatibility.

For Developers

The new URI-type strings may be passed into the existing APIs such as `CreateFromEnv` and `CreateFromFile`. There is also a new `CreateFromBuiltinConfig`. No changes should be needed, but please test that users of your application may use a path containing a URI-type string rather than a path to a config.ocio file.

If you'd like to offer the built-in configs from your user interface, there is an API to access them through the new `BuiltinConfigRegistry`. Here is basic Python code to access the registry and print the strings to present in a UI:

```
registry = ocio.BuiltinConfigRegistry().getBuiltinConfigs()
for item in registry:
    # The short_name is the URI-style name.
    # The ui_name is the name to use in a user interface.
    short_name, ui_name, isRecommended, isDefault = item

    # Don't present built-in configs to users if they are no longer recommended.
    if isRecommended:
        print(ui_name)
```

Prints:

Academy Color Encoding System - CG Config [COLORSPACES v1.0.0] [ACES v1.3] [OCIO v2.1]

Academy Color Encoding System - Studio Config [COLORSPACES v1.0.0] [ACES v1.3] [OCIO v2.1]

If your application saves the path to a config and the user enters `ocio://default`, it is recommended that you don't save that as is. Instead, call `getDefaultBuiltinConfigName` to get the name of the current default. This is guaranteed to give the same results, whereas the default config could change somewhat between releases. Prepend `ocio://` in order to save it as a valid config path. Here's the Python code:

```
"ocio://" + ocio.BuiltinConfigRegistry().getDefaultBuiltinConfigName()
```

Config Archiving

An OCIO config, including its external LUT files, may now be packaged into a single file. This makes it easier to distribute configs and may slightly discourage tampering with the contents. An archived config may be created with the new `ocioarchive` command-line tool and is stored in a file with the extension `.ocioz`. It is basically a compressed ZIP archive of the config and its LUTs.

To be archivable, the external LUT files must be contained within (or below) the config's working directory. (In OCIO terminology, the "working directory" is the directory that contains the config.ocio file.)

A config is not archivable if any of the following are true:

- It contains FileTransforms with a src outside the working directory
- The search path contains paths outside the working directory
- The search path contains paths that start with a context variable
- The working directory is not set (if archiving a Config object)

Context variables are allowed but the intent is that they may only resolve to paths that are within or below the working directory. This is because the archiving function will only archive files that are within the working directory in order to ensure that if it is later expanded, that it will not create any files outside this directory.

For example, a context variable on the search path intended to contain the name of a sub-directory under the working directory must have the form `./$DIR_NAME` rather than just `$DIR_NAME` to be considered archivable. This is imperfect since there is no way to prevent the context variable from creating a path outside the working dir, but it should at least draw attention to the fact that the archive would fail if used with context vars that try to abuse the intended functionality.

All files within or below the working directory that end with an extension corresponding to a LUT type supported by OCIO are added to the archive. This approach was taken since the goal was for the archive to work with any combination of context variables that the original config works with. Files with other extensions or no extension are not archived.

For Users

Wherever you are able to provide a file path to a config, you may now provide a path to a .ocioz file. For example, you may use this for the OCIO environment variable.

The `ocioarchive` command-line tool will allow you to archive a config or to expand an existing archive. It will also allow you to list the contents of an archived config. Run `ocioarchive --help` to get a print-out of the various options.

The print out from the `ociocheck` command-line tool now includes a new line-item indicating whether the config is archivable.

For Developers

The new .ocioz files may be passed into the existing APIs such as `CreateFromEnv` and `CreateFromFile`. No changes should be needed, but please test that users of your application may use a path containing a .ocioz file rather than a path to a config.ocio file.

The Config class has new `isArchivable` and `archive` methods. There is also an `ExtractOCIOZArchive` function.

Abstracting the Source of External LUT Files

The new ConfigIOProxy class allows the calling program to supply the config and any associated LUT files directly, without relying on the standard file system. This opens the door to expanded ways in which OCIO may be used.

The new config archiving feature was implemented using this mechanism.

For Developers

Please refer to the `ConfigIOProxy` class. By implementing the `getLutData`, `getConfigData`, and `getFastLutFileHash` methods, you have control over how the config is provided to OCIO. No file system access to a config is required.

The `CreateFromConfigIOProxy` factory allows for the creation of a Config object from a ConfigIOProxy object.

Converting To or From a Known Color Space

An OCIO config defines its own self-contained universe of color spaces. But there are not any requirements for color spaces which must always be included or how they must be named. This poses difficulties for many applications which need to convert to or from certain known standard color spaces. For example, a renderer might have a physical sun and sky model which produces colors in a CIE space and it needs to convert those into the rendering space defined by a user's custom OCIO config. Or an application may use an SDK to debayer images from a digital cinema camera. The SDK produces images in a specific color space which then needs to be processed into something viewable through a user's custom OCIO config.

For Developers

OCIO v2 introduced the Interchange Roles to help address this problem but these had previously been optional and are unlikely to be included in OCIO v1 configs (although it would be perfectly legal to add them).

OCIO 2.2 introduces the new functions `GetProcessorToBuiltinColorSpace` and `GetProcessorFromBuiltinColorSpace` that will allow you to convert to or from any of the color spaces in the built-in Default config (this is currently the ACES CG config described above). This built-in config includes common spaces such as "Linear Rec.709 (sRGB)", "sRGB - Texture", "ACEScg", and "ACES2065-1".

If the source config defines the necessary Interchange Role (typically `aces_interchange`), then the conversion will be well-defined and equivalent to calling `GetProcessorFromConfigs` with the source config and the Built-in config

However, if the Interchange Roles are not present, heuristics will be used to try and identify a common color space in the source config that may be used to allow the conversion to proceed. If the heuristics fail to find a suitable space, an exception is thrown. The heuristics may evolve, so the results returned by this function for a given source config and color space may change in future releases of the library. However, the Interchange Roles are required in config versions 2.2 and higher, so it is hoped that the need for the heuristics will decrease over time.

The current heuristics should work on any config (including an OCIO v1 config) that was generated by editing one of the ACES configs or any config that uses one of the following as its reference space:

- ACES2065-1
- ACEScg
- Scene-linear Rec.709 (sRGB)
- Scene-linear Rec.2020
- Scene-linear P3-D65

And that has a color space either for any of the above spaces or for an sRGB texture space that has "sRGB" (case-insensitive) in its color space name or one of its aliases.

Note that the heuristics create a Processor and evaluate color values that must match within a certain tolerance. No color space is selected purely based on its name alone. If the heuristics fail to find a recognized color space, an exception is thrown.

Making the interchange roles required for config versions 2.2 or higher

For Users

Users were surveyed during the OCIO 2.2 development process as to whether the Interchange Roles should become mandatory. The response was overwhelmingly in favor of doing this, largely because it allows robust interchange of color spaces between configs or to external known standard color spaces.

Therefore, as described in the previous section, for config files of version 2.2 or higher, it is mandatory to define the `aces_interchange` role. If the config includes display color spaces, the `cie_xyz_d65_interchange` role is also required.

Note that the `cie_xyz_d65_interchange` is only used in connection with display color spaces (that is, with the display-referred connection space). It is not used for scene-referred color spaces, and indeed it is an error if a scene-referred space is assigned to that role.

The `ociocheck` command-line tool has been updated to make these checks. In addition, its reporting on other roles has been modified to be more lenient regarding roles which are no longer considered essential.

For Developers

The Config::validate method will log an error if the Config object does not meet these requirements. Note that an exception is not thrown since it was felt that the Config's `upgradeToLatestVersion` method must always produce a valid config.

Determining if a Color Space is Linear

There have been many requests from developers that would like a standard way to determine if a color space is linear, since this impacts what sort of processing is suitable. OCIO v2 introduced a new `encoding` attribute for color spaces which contains this information. However, this is optional and may not be set by all config authors. And it won't be present in OCIO v1 configs, which are still widely used.

For Developers

OCIO 2.2 adds a new `isColorSpaceLinear` method to the Config class which may be used for this purpose.

Note that since OCIO has both a scene-referred and a display-referred reference space, the method also takes a ReferenceSpaceType enum to indicate which reference space the linearity determination is with respect to. Typically developers will set this to `REFERENCE_SPACE_SCENE`.

The following algorithm is used to make the determination:

- If the color space `isdata` attribute is true, return false.
- If the reference space type of the color space differs from the requested reference space type, return false.
- If the color space's encoding attribute is present, return true if it matches the expected reference space type (i.e., "scene-linear" for `REFERENCE_SPACE_SCENE` or "display-linear" for `REFERENCE_SPACE_DISPLAY`) and false otherwise.
- If the color space has no `to_reference` or `from_reference` transform, return true.
- Evaluate several points through the color space's transform and check if the output only differs by a scale factor (which may be different per channel, e.g. allowing an arbitrary matrix transform, with no offset).

Note that the last step is a heuristic that may or may not be accurate. However, note that the `encoding` attribute takes precedence and so config authors have the ultimate control over the linearity determination.

Getting a Processor for a NamedTransform

For Developers

A new config object was introduced in OCIO v2 called Named Transforms. These are used when there is a need to apply a mathematical function which is not a conversion between two specific color spaces. The most common example is applying a transfer function curve to convert linear data to non-linear, or vice-versa.

The new ACES configs include Named Transforms, so it is important for application developers to start supporting this type of config object. The preferred method for doing so is to add a new tool, similar to FileTransform that applies a Named Transform.

What is new in OCIO 2.2 is that the code for applying these is now simpler with the introduction of several new getProcessor methods on the Config class that will return a Processor directly from a NamedTransform object.

In addition, the NamedTransform class has a GetTransform method that returns a (regular) Transform object for a given direction. It will create the transform from the inverse direction if the transform for the requested direction is missing.

Circular OCIO / OIIO Build Dependency Solution

A long-standing complaint has been regarding the circular build dependency between OCIO and OpenImageIO. This is due to the fact that OIIO wants to use OCIO for color management and OCIO wants to use OIIO in its command-line tools `ociconvert`, `ociolutimage`, and `ociodisplay` for reading and writing image files. These tools will not be built if OIIO is not available when configuring the build.

Furthermore, some package installers will not install these command-line tools due to the dependency on OIIO.

By default, OCIO will now build these tools with OpenEXR rather than relying on OIIO.

For Users

If you have a version of OCIO that was not compiled with tools such as `ocioconvert` and you want to use OCIO to process images, you could try using OpenImageIO's `oiiotool`. (Although note that `ocioconvert` has a few features that are not in `oiiotool`, such as GPU processing support.) Similarly if you have `ocioconvert`, but it is compiled with OpenEXR rather than OpenImageIO, you may use `oiiotool` to convert other image file formats to/from OpenEXR.

If you want to use `oiiotool` but it does not support a particular type of conversion, you may be able to use `ociowrite` to export a CTF file and then use that with the `--ociofiletransform` option in `oiiotool`.

For Developers

In OCIO 2.2, by default, the build will now use OpenEXR rather than OpenImageIO for the command-line tools that read or write images. This will limit the functionality of the aforementioned command-line tools to only working with OpenEXR files. If you want support for more file formats in these tools, you will still need to have OIIO available when building OCIO and set the CMake variable `-D OCIO_USE_OIIO_FOR_APPS=ON`.

Miscellaneous Improvements

Here are some other improvements in OCIO 2.2:

- Support for more types of ICC Monitor Profiles – All of the parametric curve types are now supported.
- New hash function for calculating cache IDs – The md5 algorithm has been replaced with xxhash, which provides a considerable speed-up for various operations. The APIs that return cache ID strings will obviously return different strings now, but please note that these are not guaranteed to be unchanged across releases. (The 128-bit version of xxhash was used, which is the same length as for md5.)
- The command-line tools `ocioconvert`, `ociowrite`, and `ocioperf` now support using an inverse DisplayViewTransform.
- Add DisplayViewTransform and NamedTransform support to Baker.
- Several new Built-in Transforms are available for version 2.2 config files, including ARRI LogC4.
- Preliminary support for ACES Metadata File (AMF) – A prototype Python tool has been added named `pyocioamf` that converts an AMF file into the OCIO native transform format CTF. It uses a prototype ACES Reference config file that is serving as a database of ACES Transform IDs for interpreting the AMF file.
- Support for PyPI installation from source rather than pre-built binaries.

5.6.12 Release Notes

For more detail, please see the GitHub release pages:

OCIO 2.2.0

OCIO 2.2.1

5.6.13 OCIO 2.1 Release

Timeline

OpenColorIO 2.1 was delivered in August 2021 and is in the VFX Reference Platform for calendar year 2022.

5.6.14 New Feature Guide

ACES 1.3 Gamut Compression

ACES 1.3 introduced a scene-referred gamut compression algorithm that compresses colors captured by a camera into the AP1 gamut. This avoids artifacts that may occur downstream (such as in the ACES Output Transforms) for scene elements such as colored lights. This supersedes the earlier "Blue Light Artifact Fix" LMT.

ACES RGC User Guide

ACES RGC Implementation Guide

The implementation includes a BuiltinTransform for the ACES Reference Gamut Compression. Here's what that looks like in Config YAML:

```
!<BuiltinTransform> {style: ACES-LMT - ACES 1.3 Reference Gamut Compression}
```

There is also a FixedFunctionTransform for the underlying gamut compression algorithm. As described in the links above, it takes seven parameter values:

```
[ Limit_Cyan, Limit_Magenta, Limit_Yellow, Threshold_Cyan, Threshold_Magenta, Threshold_
↪Yellow, Roll-off ].
```

Here is how that looks in Config YAML with the parameters set for the ACES 1.3 Reference Gamut Compression:

```
!<FixedFunctionTransform> {style: ACES_GamutComp13, params: [1.147, 1.264, 1.312, 0.815,␣
↪0.803, 0.88, 1.2]}
```

And here's how it looks in a CTF file:

```
<FixedFunction inBitDepth="32f" outBitDepth="32f" style="GamutComp13Fwd" params="1.147 1.
↪264 1.312 0.815 0.803 0.88 1.2" />
```

Config authors: These new transforms may be used in configs with version 2.1 or higher and CTF files with version 2.1 or higher.

OpenFX OCIO plug-ins

A framework to support OpenFX plug-ins has been added and example source code may be found in the vendor/openfx directory. The initial set consists of two plug-ins, one for applying a ColorSpaceTransform and another for applying a DisplayViewTransform.

Some screenshots of the UI may be found in PR #1371.

End users: There are currently no pre-built executables available, so you will need to compile these yourself.

Developers: Set the CMake variable `-D OCIO_BUILD_OPENFX=ON` to build the plug-ins.

Support for PyPI (pip install)

Python wheel generation has been added and support for the Python Package Index (PyPI). This allows you to easily install the OCIO Python bindings without needing to build from source or use one of the OS-specific package managers. The command is simply:

```
pip install opencolorio
```

Support for emitting Open Shading Language (OSL)

The OCIO GPU Renderer may now emit shaders in Open Shading Language format. There is a new GpuLanguage enum: `LANGUAGE_OSL_1`.

Note that the OCIO library does not need to be compiled against the OSL library but running the OSL unit tests does require OSL and OIIO. These will be built automatically if CMake is able to find them or you set `-D OSL_ROOT` to indicate the installed location.

Currently, Lut1D and Lut3D Transforms are not supported and will throw an exception if used. All other transforms are supported. However, dynamic properties are currently locked to their initial values and may not be adjusted.

Support for Apple Metal Shading Language and OpenGL ES

The OCIO GPU Renderer may now emit shaders in these other new languages. The new GpuLanguage enums are:

- `GPU_LANGUAGE_MSL_2_0`
- `GPU_LANGUAGE_GLSL_ES_3_0`
- `GPU_LANGUAGE_GLSL_ES_1_0`

Test suite for the Academy/ASC Common LUT Format (CLF)

The set of CLF test files was expanded and a set of Python scripts was added under `share/clf` which were developed by the ACES Common LUT Format Implementation Working Group. The scripts allow use of the OpenColorIO renderers to process a sample image through each CLF file in the test suite and then compare the reference images to a set of corresponding images from another CLF implementation to see if the differences are within the tolerance allowed by the CLF Implementation Guide.

Imath 3 support

OCIO and its CI system now support Imath 3 for the Half data type dependency.

Noteworthy API updates

getColorSpaceFromFilePath

`Config::getColorSpaceFromFilepath()` is now the proper way to extract a color space from a file path, regardless of whether the config is version 1, or version 2 or higher. It works regardless of whether the config has FileRules defined. Therefore, the older `Config::parseColorSpaceFromString()` method (which does not work with FileRules) is officially deprecated.

Note that `Config::getColorSpaceFromFilepath()` always returns a color space, regardless of whether the config's `strictparsing` attribute is true or false. This was a request from app developers that always need a default (which is a common scenario). Please see Issue #1398 for more details on how to take special action if `strictparsing` is true.

getDefaultView

There is a new `Config::getDefaultView(display, colorspaceName)` method that takes advantage of the ViewingRules feature that was introduced in OCIO 2.0. It returns the default view that is most appropriate for a given color space. If the color space is known, developers should call this rather than `Config::getDefaultView(display)`.

The motivation is that the best view transform to use varies with the color space. For example, a scene-referred color space typically wants some kind of tone-map such as an ACES Output Transform. However, a display-referred color space will look wrong with such a view transform since it has already been tone-mapped.

One scenario where this new method is ideal is when an application needs to generate thumbnails for a large number of images that are in a variety of color spaces. Note that the config must have the Viewing Rules set in order to return different views, but if the rules are not set, it will still return the same result as `Config::getDefaultView(display)`, so it may be used with either v1 or v2 configs.

5.6.15 Release Notes

For more detail, please see the GitHub release pages:

OCIO 2.1.0

OCIO 2.1.1

OCIO 2.1.2

OCIO 2.1.3

5.6.16 OCIO 2.0 Release

OpenColorIO v2 is a major update, three years in development, and contains a large number of new features. This section describes the new features and their relevance for both config authors and application developers.

Timeline

OpenColorIO v2 was proposed to the community at SIGGRAPH 2017 and reached "feature complete" at SIGGRAPH 2020. The official 2.0.0 release was made in January 2021 after a period of stabilization and refinement. OCIO v2 is in the VFX Reference Platform for calendar year 2021.

Demo Config

There is a config file that illustrates many of the new OCIO v2 features described below. This is available in the configs section of the documentation as *ocio-v2_demo*.

5.6.17 New Feature Guide

New GPU Renderer

The GPU renderer in OCIO v1 baked the operators into a form that was easier to use on the GPUs of the day. It did not fully match the CPU renderer and was not suitable for rendering final frames. The GPU renderer has been completely replaced for OCIO v2 and now computes pixels using the same operators as the CPU.

Config authors: You should experience a faithful match between CPU and GPU results.

Developers: The GPU API needed to be completely redone in order to support uploading the necessary textures to the GPU. The new API is more powerful and allows more options for customizing the GPU renderer to meet the needs of a wider variety of applications.

Enhanced CPU Renderer

The CPU renderer has been improved and uses more vectorized (SSE) instructions, support for integer pixel buffers, and improved optimization.

Config authors: You should notice improved performance (up to 20x) in some cases.

Developers: A compliation setting allows you to turn on/off SSE usage. There is a new command-line tool `ociоperf` to test performance on a given system with a given color space conversion.

Support for Integer Pixel Buffers

OCIO v1 only supported floating-point pixel buffers and only supported in-place processing. OCIO v2 supports integer pixels and does not require in-place processing. There is also support for a variety of pixel layouts (e.g. RGB and BGR). Using the integer buffers where possible not only allows optimization of memory traffic, it also allows OCIO to do additional optimizations.

Config authors: No impact.

Developers: Update your apply calls to use integer buffers where possible. This is one of the biggest performance wins for CPU processing.

Improved Optimization

OCIO v2 is able to do a lot more optimization of a Processor in preparation for the pixel apply call. There is also now an environment variable that may be used to provide very fine-grained control over the optimization settings.

Config authors: You may now override optimization settings to troubleshoot, if needed via the `OCIO_OPTIMIZATION_FLAGS` environment variable.

Developers: There is now an extra step between getting a Processor and calling apply(). You need to get a CPUProcessor or GPUProcessor and this is where you may specify custom optimization settings. Note that the environment variable allows users to override these settings.

Inversion of all Operators

In OCIO v1, 3D-LUTs could not be inverted and this caused some conversions to fail. In OCIO v2, all transforms may be inverted.

Config authors: If you were building separate 3D-LUTs for both the from- and to_reference directions, you may not need to do that. However, if you are applying sophisticated gamut mapping, the separate transform is probably still worth making.

Developers: You don't need to worry about some color conversions failing.

New Operators to Support ACES

Users have sometimes run into issues with the ACES config when processing extremely bright values and finding the results don't match the official Academy CTL results. In OCIO v2, new operators have been added to facilitate an exact match to ACES Output Transforms. Note that the new v2 ACES OCIO Config is still under development.

Config authors: If you use ACES Output Transforms, please keep an eye out for the v2 ACES Config that will leverage built-in ACES transforms.

Developers: The upcoming v2 ACES Config will allow your application to have very accurate ACES performance on both CPU and GPU.

Built-in Color Transforms

OCIO v2 has the ability to generate a set of commonly used transforms on demand in memory, avoiding the need for external LUT files. This means that some configs may be built with no need for any external files. The current set of BuiltInTransforms includes the ACES Color Space Conversion transforms as well as the components needed for the ACES Output Transforms. (In OCIO v2, the ACES Output Transforms are factored into a View Transform and a Display Color Space.)

Config authors: You will be able to simply use a built-in transform rather than creating your own versions of common color spaces.

Developers: It is possible to create configs that don't require external LUT files and are thus more robust (e.g. render farms, cloud processing).

Support for Academy/ASC CLF

OCIO v2 has a very thorough implementation of the Common LUT Format (CLF) released with ACES 1.2. This LUT format offers many advantages over existing formats. A new command-line tool `ociomakeclf` will convert any of the LUT formats supported by OCIO into CLF format. Also, it is able to insert conversions from and to ACES2065-1 before and after the LUT in order to make an ACES-compliant LMT that may be used in an ACES Metadata File.

Config authors: The features of CLF may allow you to write smaller and/or more accurate LUT files for your configs. If you were saving transform components in separate files, you may combine them into a single file, if desired.

Developers: You may now add CLF support to your application. Note that this is necessary in order to obtain ACES Logo Certification. Note that CLF has much better metadata support compared to previous LUT formats and so apps should expose these fields to users.

Serialization of all OCIO transforms as CTF

The Autodesk Color Transform Format (CTF) is able to serialize all OCIO transforms into an XML format that is a superset of Academy/ASC CLF. This is very useful for troubleshooting. It also opens up new workflow possibilities. The new `ociowrite` command-line tool will serialize an OCIO Processor object to a CTF file. Similarly there is a write method on the GroupTransform class.

Config authors: You may use CTF to store a chain of arbitrary OCIO transforms to an XML file for use in a config or to send as a self-contained file.

Developers: A given OCIO Processor may be easily serialized and restored.

Access to all transforms from the public API

The public API now allows access to everything OCIO is able to load/read, including LUT entries. Also, an OCIO Processor may now be converted into a GroupTransform. This opens up new ways of using OCIO.

Config authors: No impact.

Developers: Even if you have your own color rendering engine, you may now use OCIO to read and write the many LUT formats it supports, including the Common LUT Format (CLF).

Display-referred Connection Space

There is now a second reference space in OCIO. The original reference space is typically a scene-referred color space and the new space is intended to be for a display-referred color space. This means that the conversion from a scene-referred space to a display space may be broken down into a view transform plus a display color space. There are new config sections for `view_transforms` and `display_colorspaces`.

Config authors: Break down your Views into a view transform and display color space. Having a separate display color space faciitates direct conversion from one display to another without needing to convert back to the scene-referred reference space.

Developers: No impact. The new display color spaces will appear in menus with other color spaces by default.

Shared Views

It is now possible to define a View and reuse it for multiple displays. There is a new `shared_views` section of the config file.

Config authors: Make your configs easier to read and maintain by using shared views.

Developers: No impact.

Support for ICC Monitor Profiles

OCIO v2 is able to read basic ICC monitor profiles. Also a new virtual display object in the config allows a config author to define how OCIO may instantiate a new display and views from a user's ICC monitor profile.

Config authors: Add a `virtual_display` to your config to enable a user to use the ICC profile for their monitor. Note that if you do not add the virtual display, applications will not be able to instantiate an ICC profile with your config.

Developers: There is new SystemMonitor code to use to ask OCIO to instantiate a new display and views from an ICC profile. This is also able to query the operating system on Mac and Windows to obtain the user's current profile.

A Categories Attribute for Color Spaces

A new attribute called `categories` has been added to color spaces. The goal is to allow applications to filter the complete list of color spaces down to only show users the ones needed for the task at hand. For example, when choosing a working space, it may not be ideal to show all the color spaces in the config. The list of color spaces is filtered by both a "user-defined set" and an "application-defined" set of categories. For example, the user-defined set could include categories for which department (lighting, comp, etc.) and the artist experience level (e.g. basic or advanced). The application-defined set could include categories for different types of menus, for example working spaces vs. file IO.

Config authors: Add the categories attribute to help applications shorten their menus to only include the appropriate color spaces for various tasks. Configure your app configuration scripts to set the `OCIO_USER_CATEGORIES` env var with the appropriate categories.

Developers: Use the Menu Helpers classes to build your application color space menus to take advantage of this feature. Publish the category string you use for each of your color space menus so config authors may use them in their configs.

Encoding Attribute

There is a new attribute called `encoding` that may be used to indicate the type of encoding used for a color space. The encoding options are `scene-linear`, `display-linear`, `log`, `sdr-video`, `hdr-video`, and `data`. This is useful to applications since image processing algorithms often need to know the encoding for optimium results. The encoding may also be used in the viewing rules to filter views based on the color space. The encoding attribute may be used in combination with the categories. For example, an application may ask for the list of all color spaces with a category "working-space" and an encoding "scene-linear".

Config authors: Set the encoding attribute on your color spaces to help applications know how to process images in that space better. Also, use the encoding in viewing rules to allow applications to filter the views to be appropriate for a given color space.

Developers: Knowing the encoding of a color space may allow you to optimize your image processing algorithms.

Inactive color space list

There is now an `inactive_colorspaces` list in the config and a corresponding environment variable. This allows config authors to keep color spaces in a config but prevent them from appearing in application menus.

Config authors: This allows you to remove color spaces you don't want users to have access to. This may also be set via the `OCIO_INACTIVE_COLORSPACES` environment variable.

Developers: These color spaces will not show up in the normal list of color spaces, however you may still use them as arguments to getProcessor. (For example if your application has assets that use an earlier version of the config where those spaces were active.) The Menu Helpers classes show how to deal with temporarily adding an inactive color space to menus when it is necessary.

Color space Aliases

There is a new `aliases` attribute that allows the config author to define a set of alternate names for a color space. For example, the official name may be a user-friendly name for the UI and a shorter name that is intended for embedding in file paths may be declared as an alias.

Config authors: This avoids the need to add duplicates of color spaces simply to have an alternate name. It may also be used to handle deprecated color space names.

Developers: There is also a new getCanonicalName method on the Config that may be used to resolve aliases to the official name. Before saving a color space name, it may be desirable to convert to the canonical name.

Hierarchical menus

The config has a new family_separator attribute that specifies a character to be used in the `family` attribute to break strings down into a hierarchy. The Menu Helpers is able to generate hierarchical menus based on this.

Config authors: Use the family attribute to help applications organize long color space lists better.

Developers: Use the Menu Helpers classes to build your application color space menus to take advantage of this feature.

Color Picker Helper

There are Mixing Helpers classes that show how to implement a color picker that works well with scene-linear data. This facilitates making UI sliders for linear values and also doing sensible RGB to HSV conversions with float values outside [0,1].

Config authors: No impact.

Developers: This simplifies making scene-linear friendly color pickers.

File Rules

The File Rules allow a config to specify how to assign a default color space to a file based on the path using glob or regex pattern matching. This opens up new workflows since it is no longer necessary to embed a color space name into the path.

Config authors: You may not need to embed a color space name into your paths anymore. You may be able to rely on better default file handling among various applications.

Developers: Implement support for the new file rules. Also, if your application honored "strictparsing: true" mode in OCIO v1, the code for doing this has changed in v2. It is now always possible to obtain from OCIO a valid default color space for a file.

Viewing Rules

The Viewing Rules allows a config author to specify which Views in a display are appropriate for a given color space. It also makes it possible to have the default view be a function of the color space.

Config authors: Set the viewing rules to enable friendlier application behavior.

Developers: When asking for the list of views for a display, use the new API that allows passing in the color space being viewed. The first view in the list is the most appropriate one for that color space. This is useful as the default view for the first time an image or asset is viewed or for generating sensible proxy or thumbnail images.

Dynamic Properties

Certain transforms now support dynamic properties which are parameters that may be adjusted even after a transform has been converted into a Processor. This is useful especially when users are making live updates, for example when adjusting the exposure or gamma of an image in a viewport. On the GPU, these are mapped to uniforms.

Config authors: No impact. Dynamic properties are not exposed in configs, which are locked representations of a color pipeline.

Developers: Expose dynamic properties in situations where the user is creating a transform with editable values or when editing exposure or gamma controls on a viewing pipeline for improved performance. See the DisplayViewHelpers application helpers code or the unit tests for examples. Note however that transforms included in a config should be considered locked.

New Transforms for Building Looks

ASC CDL transforms are easy to edit but are not very powerful, whereas a Lut3D is very powerful, but difficult to edit and understand what it does. OCIO v2 introduces some new transforms that fall in a middle ground – they are more powerful than a CDL but are also parametrically adjustable and easy to read. The new transforms are for Primary adjustments, fine adjustments to Tone reproduction, and spline-based RGB curves. The new transforms make use of dynamic properties to facilitate live interactive adjustments on the CPU and GPU.

Config authors: You may find these new transforms useful when building Look Transforms. When used in a config, the parameters may be considered locked (same as a CDL).

Developers: You may want to expose editing functionality for these transforms and support their dynamic properties. Note however that once a transform is included in a config, it should be considered locked.

Providing an Interchange Mechanism Between Configs

In OCIO v1, there was no way to convert an image or asset in a color space from one config into a color space from a different config. This presented a serious challenge for some workflows. In OCIO v2 there are new APIs that enable this conversion. However, it requires the config author to implement new roles called aces_interchange and cie_xyz_d65_interchange.

Config authors: Please implement these roles in your configs.

Developers: This feature may open up long awaited workflows for you.

Named Color Transforms

There is a new `named_transforms` section of the config that may be used to define transforms that the config author wants to make available but which are not actually a color space, look, or view. For example, a gamut mapping transform or a utility curve transform.

Config authors: These may provide a useful alternative to color spaces.

Developers: These new transforms do not appear in color space menus by default. So action is needed to make them available to users. This could be done by adding a new Named Transform tool alongside an existing Color Space Transform tool, for example.

Processor Caching

In OCIO v2 there is now a cache for Processors. The caching system is aware of context variables and will detect if a variable has been changed.

Config authors: No impact.

Developers: This may facilitate various options such as realtime playback of timelines that leverage context variables.

New Description and Name Attributes

A new `name` attribute has been added to many of the OCIO transforms to provide some additional labeling options in a config file. The `description` attribute has been added to Views to allow similar description strings as are used in color spaces. There is also a `name` attribute at the top level of the config itself.

Config authors: These may prove useful. Consider updating the name attribute of the config as the config file is versioned.

Developers: Consider exposing the description string for both color spaces and views.

5.6.18 Changes from v1

DisplayTransform

The decision was made to refactor DisplayTransform to make it easier to use and easier to invert. The functionality of the DisplayTransform is now in the LegacyViewingPipeline class in src/OpenColorIO/apphelpers. The original DisplayTransform class has been removed. There is a new DisplayViewTransform available that now supports inversion.

Config authors: No impact.

Developers: The DisplayViewTransform, along with ColorSpaceTransform are the two key pieces of functionality to expose to users. If you were using the original DisplayTransform, update to LegacyViewingPipeline for viewports. But you should also consider exposing DisplayViewTransform to users as a tool for baking in (or inverting) a display + view.

Clamping

In OCIO v1, the exponent transform was used to implement the ASC CDL and it had unusual clamping behavior where it would clamp negative values *except* if the power was 1. The decision was made to add new transforms that provide more clamping options. A style attribute has been added to the ASC CDL and Exponent Transforms that allow a variety of negative-handling options to be selected. For v1 configs, the original exponent behavior is used, but in v2 configs, the new operators are used. Also, in OCIO v1, the optimization process sometimes changed whether a transformation clamped or not. In v2, the optimized transforms more closely follow the clamping behavior of the original.

Config authors: Verify the clamping behavior in your configs and adjust the style arguments as desired when upgrading configs to v2.

Developers: There should be fewer differences in behavior due to changes in clamping based on parameter value changes or optimization changes.

Strict Parsing Mode

The code for implementing "strictparsing: true" mode has changed from v1.

Config authors: If you use this mode, verify that your applications support it as you expect.

Developers: See the File Rules API for more info.

Default Role

In OCIO v1, the default role was sometimes used as a fallback in general cases where a color space could not be found. This is no longer the case.

Config authors: If you relied on this behavior, please verify your configs. Note that this behavior may have hid errors that existed in your configs.

Developers: No impact.

Context Variable Changes

In a v2 config, it is now illegal to use the context variable tokens '$' and '%' in color space names (in other words, it is illegal to use them if they are not actually context variables).

Config authors: Please do not use these characters except for context variables.

Developers: No impact.

Allocation Variables

The color space allocation variables are not used by the new GPU renderer. However, they are still used by the ociobakelut utility and if an application requests the legacy GPU shader.

Config authors: If you don't care about baking and you are using applications that use the new GPU renderer, you don't need to set the allocation variables on color spaces anymore.

Developers: Update your applications to use the new GPU renderer.

Nearest Interpolation Method

OCIO v1 implemented a "nearest neighbor" intepolation method for Lut1D and Lut3D which is faster than the default linear interpolation but not as accurate. Since processing of integer pixel buffers was not supported in v1, the "nearest" method may have been used by some to speed up applying a Lut1D to integer source material. OCIO v2 does not implement a separate "nearest" method, and FileTransforms that request this style will receive "linear" instead. However, OCIO v2 does support processing of integer pixel buffers directly and this provides a big speed-up for this use-case without requiring "nearest" mode (i.e., a look-up is done without interpolation for Lut1D ops when the inputs are integers).

Config authors: Please be aware of this change.

Developers: To benefit from this performance improvement, please call OCIO directly on integer pixel buffers rather than converting them to float buffers first.

Color Space Conversion No-ops

In OCIO v1, a ColorSpaceTransform from color space A to color space A (i.e., using the same color space for source and destination) was not necessarily a no-op. In OCIO v2, this is a no-op. In other words, all color spaces may be considered to be in an equality group with themselves.

Config authors: Please be aware of this change.

Developers: No impact.

CDLTransform XML read/write

In OCIO v1, the CDLTransform had getXML and setXML methods. In OCIO v2, the read/write/metadata handling has been refactored to be more like other formats. CreateFromFile is the replacement for setXML and the write method on the GroupTransform class is the replacement for getXML. The FormatMetadata class provides extensive metadata access.

Config authors: No impact.

Developers: Please update your code to use the new methods.

Version Handling

In OCIO v1, the YAML parser would warn if the config version was higher than the library version but continue to try to read it. In OCIO v2, if the config version is higher than the library it will immediately throw an exception.

Config authors: Please be aware of the change.

Developers: No impact.

Bypassing Data Spaces

In OCIO v1, if a color space set its isdata attribute to true, a ColorSpaceTransform would be a no-op. The DisplayTransform also bypassed most of its processing. In OCIO v2, data spaces are still bypassed by default. However, the application developer may now use a new argument to specify that data spaces should be processed.

Config authors: Please be aware of the change.

Developers: This may be useful, for example, when displaying data color spaces on an HDR monitor.

5.6.19 Release Notes

For more detail, please see the GitHub release pages:

OCIO 2.0.0

OCIO 2.0.1

OCIO 2.0.2

OCIO 2.0.3

OCIO 2.0.4

OCIO 2.0.5

5.7 API

5.7.1 Applications Helpers

ColorSpaceMenuHelpers

Python

PyOpenColorIO.ColorSpaceHelpers.**AddColorSpace**(*config: PyOpenColorIO.PyOpenColorIO.Config*, *name: str*, *transformFilePath: str*, *categories: str = ''*, *connectionColorSpaceName: str*) → None

Add a new color space to the config instance. The output of the userTransform must be in the specified connectionColorSpace.

Note: If the config does not already use categories, we do not add them since that would make a big change to how existing color spaces show up in menus.

class PyOpenColorIO.**ColorSpaceMenuParameters**

Parameters controlling which color spaces appear in menus.

The ColorSpaceMenuHelper class is intended to be used by applications to get the list of items to show in color space menus.

The ColorSpaceMenuParameters class is used to configure the behavior as needed for any given menu. Here is the algorithm used to produce a list of "items" (or strings) that will appear in a menu:

1) Use setRole to identify a role that controls a given menu. If the config has this role, then only that color space is returned. The name is set to the color space name, the UIName is presented as "<role name> (<color space name>)". It may be useful for the application to then grey-out the menu or otherwise indicate to the user that the value for this menu is not user selectable since it was pre-determined by the config. If the config does not have that role, the algorithm continues to the remaining steps.

2) The IncludeColorSpaces, SearchReferenceSpaceType, and IncludeNamedTransforms parameters are used to identify a set of items from the config that are potential candidates for use in the menu, as follows: - IncludeColorSpaces: Set to true to include color spaces in the menu. - SearchReferenceSpaceType: Use this to control whether the menu should include all color spaces, only display color spaces, or only non-display color spaces. - IncludeNamedTransforms: Set to true to include named transforms in the menu.

3) The set of items from step 2 is then filtered in step 3 using the following parameters: - AppCategories: A list of strings specified by the application based on the purpose of the menu. For example, if the menu is used to select a color space for importing an image, the application might specify the 'file-io' category, whereas if it is to select a working color space, it might specify the 'working-space' category. Application developers should document what strings they are using for each menu so that config authors know what categories to use in their configs. Alternatively, an application could let advanced users customize the string to use for a given menu in the application. - Encodings: A list of strings used to further refine the items selected from the AppCategories. For example, an application might specify 'working-space' as the category and then specify 'scene-linear' as the encoding to only use items that have both of those properties (e.g., only select scene-linear working color spaces). - UserCategories: A list of strings specified by the end-user of the application. OCIO will check for these strings in an environment variable, or they may be passed in from the application. - TreatNoCategoryAsAny: By default, color spaces (or named transforms) that have no categories are handled as if they had any of the categories. Config authors that want to hide color spaces without categories should either put them in the inactiveColorSpaces list or add a category that will never be searched for (e.g., "invisible" or "hidden"). App developers may set this option to false to hide items without any categories.

Basically the intent is for the filtering to return the intersection of the app categories, encoding, and user categories. However, some fall-backs are in place to ensure that the filtering does not remove all menu items. Here is the detailed description:

3a) The items from step 2 are filtered to generate a list of appItems containing only the ones that contain at least one of the AppCategories strings in their "categories" property and one of the encodings in their "encoding" property. If this list is empty, an attempt is made to generate a non-empty appItems list by only filtering by AppCategories. If that is empty, an attempt is made to only filter by Encodings.

3b) The items from step 2 are filtered to generate a list of userItems containing only the ones that have at least one of the UserCategories strings in their "categories" property.

3c) If both appItems and userItems are non-empty, a list of resultItems will be generated as the intersection of those two lists.

3d) If the resultItems list is empty, the appList will be expanded by only filtering by AppCategories and not encodings. The resultItems will be formed again as the intersection of the appItems and userItems.

3e) If the resultItems is still empty, it will be set to just the appItems from step 3a.

3f) If the resultItems is still empty, it will be set to just the userItems.

3g) If the resultItems is still empty, the items are not filtered and all items from step 2 are returned. The rationale is that if step 2 has produced any items, it is not acceptable for step 3 to remove all of them. An application usually expects to have a non-zero number of items to display in the menu. However, if step 2 produces no items (e.g. the application requests only named transforms and the config has no named transform), then no items will be returned.

4) **If IncludeRoles is true, the items from step 3 are extended by including an item for each role. The name is set to the role name, the UIName is presented as "<role name> (<color** space name>)", and the family is set to "Roles".

5) If AddColorSpace has been used to add any additional items, these are appended to the final list.

ColorSpaceMenuParameters(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuParameters, config: PyOpenColorIO.PyOpenColorIO.Config) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuParameters, config: PyOpenColorIO.PyOpenColorIO.Config, role: str = '', includeColorSpaces: bool = True, searchReferenceSpaceType: PyOpenColorIO.PyOpenColorIO.SearchReferenceSpaceType = <SearchReferenceSpaceType.SEARCH_REFERENCE_SPACE_ALL: 2>, includeNamedTransforms: bool = False, treatNoCategoryAsAny: bool = True, appCategories: str = '', encodings: str = '', userCategories: str = '', includeRoles: bool = False) -> None

addColorSpace(*colorSpace: str*) → None

Add an additional color space (or named transform) to the menu.

Note that an additional color space could be: - an inactive color space, - an active color space not having at least one of the selected categories, - a newly created color space. Will throw when creating the menu if color space is not part of the config. Nothing is done if it is already part of the menu. It's ok to call this multiple times with the same color space, it will only be added to the menu once. If a role name is passed in, the name in the menu will be the color space name the role points to.

clearAddedColorSpaces() → None

getAddedColorSpaces() → PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuParameters.AddedColorSpaceIterator

getAppCategories() → str

getConfig() → *PyOpenColorIO.PyOpenColorIO.Config*

getEncodings() → str

getIncludeColorSpaces() → bool

getIncludeNamedTransforms() → bool

getIncludeRoles() → bool

getRole() → str

getSearchReferenceSpaceType() → *PyOpenColorIO.PyOpenColorIO.SearchReferenceSpaceType*

Can be used to restrict the search using the ReferenceSpaceType of the color spaces. It has no effect on roles and named transforms.

getTreatNoCategoryAsAny() → bool

getUserCategories() → str

setAppCategories(*appCategories: str*) → None

App categories is a comma separated list of categories. If appCategories is not NULL and not empty, all color spaces that have one of the categories will be part of the menu.

setConfig(*config:* PyOpenColorIO.PyOpenColorIO.Config) → None

Config is required to be able to create a ColorSpaceMenuHelper.

setEncodings(*encodings: str*) → None

Encodings is a comma separated list of encodings. When not empty, is retricting the search to color spaces that are using one of the encodings.

setIncludeColorSpaces(*includeColorSpaces: bool = True*) → None

Include all color spaces (or not) to ColorSpaceMenuHelper. Default is to include color spaces.

setIncludeNamedTransforms(*includeNamedTransforms: bool = True*) → None

Include all named transforms (or not) to ColorSpaceMenuHelper. Default is not to include named transforms.

setIncludeRoles(*includeRoles: bool = True*) → None

Include all roles (or not) to ColorSpaceMenuHelper. Default is not to include roles. Roles are added after color spaces with an single hierarchy level named "Roles".

setRole(*role: str*) → None

If role is a valid role, other parameters are ignored and menu will contain only that role.

setSearchReferenceSpaceType(*searchReferenceSpaceType:* PyOpenColorIO.PyOpenColorIO.SearchReferenceSpaceType) → None

setTreatNoCategoryAsAny(*treatNoCategoryAsAny: bool = True*) → None

When searching for color spaces using app or user categories, treat color spaces or named transforms that have no categories as if they had any of the categories. Default is true.

setUserCategories(*categories: str*) → None

User categories is a comma separated list of categories. If OCIO_USER_CATEGORIES_ENVVAR env. variable is defined and not empty, this parameter is ignored and the value of the env. variable is used for user categories.

class PyOpenColorIO.**ColorSpaceMenuHelper**

Helper class to create menus for the content of a config.

Menu can list color spaces, roles, named transforms. Each entry has a name, a UI name, a description, and a family. Family can also be accessed as hierarchy levels; levels are created by splitting the family using the 'family separator'. Hierarchy levels are meant to be used as sub-menus.

The UI name is what is intended to be put in application menus seen by the end-user. However, please note that the UI name is not guaranteed to remain stable between releases and so if applications need to save something it should be the 'name' rather than the 'UI name'. Currently, the only difference between the 'name' and 'UI name' is for roles.

The overall ordering of items is: color spaces, named transforms, roles, and additional color spaces. The display color spaces will either come before or after the other color spaces based on where that block of spaces appears in the config. The order of items returned by the menu helper preserves the order of items in the config itself for each type of elements, thus preserving the intent of the config author. For example, if you call getName at idx and idx+1, the name returned at idx+1 will be from farther down in the config than the one at idx as long as both are of the same type. (An application may ask for only the items in one of those blocks if it wants to handle them separately.) If the application makes use of hierarchical menus, that will obviously impose a different order on what the user sees in the menu. Though even with hierarchical menus, applications should try to preserve config ordering (which is equivalent to index ordering) for items within the same sub-menu.

ColorSpaceMenuHelper*parameters:* PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuParameters → None

getDescription(*index: SupportsInt*) → str

getFamily(*index: SupportsInt*) → str

getHierarchyLevels(*index: SupportsInt*) → *PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator*

getIndexFromName(*name: str*) → int

Get the index of the element of a given name. Return (size_t)-1 name if NULL or empty, or if no element with that name is found.

getIndexFromUIName(*name: str*) → int

getName(*index: SupportsInt*) → str

Get the color space (or role) name used in the config for this menu item. Will be empty if the index is out of range.

getNameFromUIName(*name: str*) → str

Get the color space name from the UI name.

getNumColorSpaces() → int

Access to the color spaces (or roles).

getUIName(*index: SupportsInt*) → str

Get the name to use in the menu UI. This might be different from the config name, for example in the case of roles. Will be empty if the index is out of range.

getUINameFromName(*name: str*) → str

Get the color space UI name from the name.

class PyOpenColorIO.ColorSpaceMenuHelper.**ColorSpaceLevelIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator*

len(*self*) → int

next(*self*) → str

C++

class **ColorSpaceMenuParameters**

Parameters controlling which color spaces appear in menus.

The ColorSpaceMenuHelper class is intended to be used by applications to get the list of items to show in color space menus.

The ColorSpaceMenuParameters class is used to configure the behavior as needed for any given menu. Here is the algorithm used to produce a list of "items" (or strings) that will appear in a menu:

1) Use setRole to identify a role that controls a given menu. If the config has this role, then only that color space is returned. The name is set to the color space name, the UIName is presented as "<role name> (<color space name>)". It may be useful for the application to then grey-out the menu or otherwise indicate to the user that the value for this menu is not user selectable since it was pre-determined by the config. If the config does not have that role, the algorithm continues to the remaining steps.

2) The IncludeColorSpaces, SearchReferenceSpaceType, and IncludeNamedTransforms parameters are used to identify a set of items from the config that are potential candidates for use in the menu, as follows:

- IncludeColorSpaces: Set to true to include color spaces in the menu.
- SearchReferenceSpaceType: Use this to control whether the menu should include all color spaces, only display color spaces, or only non-display color spaces.
- IncludeNamedTransforms: Set to true to include named transforms in the menu.

3) The set of items from step 2 is then filtered in step 3 using the following parameters:

- AppCategories: A list of strings specified by the application based on the purpose of the menu. For example, if the menu is used to select a color space for importing an image, the application might specify the 'file-io' category, whereas if it is to select a working color space, it might specify the 'working-space' category. Application developers should document what strings they are using for each menu so that config authors know what categories to use in their configs. Alternatively, an application could let advanced users customize the string to use for a given menu in the application.
- Encodings: A list of strings used to further refine the items selected from the AppCategories. For example, an application might specify 'working-space' as the category and then specify 'scene-linear' as the encoding to only use items that have both of those properties (e.g., only select scene-linear working color spaces).
- UserCategories: A list of strings specified by the end-user of the application. OCIO will check for these strings in an environment variable, or they may be passed in from the application.
- TreatNoCategoryAsAny: By default, color spaces (or named transforms) that have no categories are handled as if they had any of the categories. Config authors that want to hide color spaces without categories should either put them in the inactiveColorSpaces list or add a category that will never be searched for (e.g., "invisible" or "hidden"). App developers may set this option to false to hide items without any categories.

Basically the intent is for the filtering to return the intersection of the app categories, encoding, and user categories. However, some fall-backs are in place to ensure that the filtering does not remove all menu items. Here is the detailed description:

3a) The items from step 2 are filtered to generate a list of appItems containing only the ones that contain at least one of the AppCategories strings in their "categories" property and one of the encodings in their "encoding" property. If this list is empty, an attempt is made to generate a non-empty appItems list by only filtering by AppCategories. If that is empty, an attempt is made to only filter by Encodings.

3b) The items from step 2 are filtered to generate a list of userItems containing only the ones that have at least one of the UserCategories strings in their "categories" property.

3c) If both appItems and userItems are non-empty, a list of resultItems will be generated as the intersection of those two lists.

3d) If the resultItems list is empty, the appList will be expanded by only filtering by AppCategories and not encodings. The resultItems will be formed again as the intersection of the appItems and userItems.

3e) If the resultItems is still empty, it will be set to just the appItems from step 3a.

3f) If the resultItems is still empty, it will be set to just the userItems.

3g) If the resultItems is still empty, the items are not filtered and all items from step 2 are returned. The rationale is that if step 2 has produced any items, it is not acceptable for step 3 to remove all of them. An application usually expects to have a non-zero number of items to display in the menu. However, if step 2 produces no items (e.g. the application requests only named transforms and the config has no named transform), then no items will be returned.

4) If IncludeRoles is true, the items from step 3 are extended by including an item for each role. The name is set to the role name, the UIName is presented as "<role name> (<color

space name>)", and the family is set to "Roles".

5) If AddColorSpace has been used to add any additional items, these are appended to the final list.

Public Functions

virtual void **setConfig**(*ConstConfigRcPtr* config) noexcept = 0

Config is required to be able to create a ColorSpaceMenuHelper.

virtual *ConstConfigRcPtr* **getConfig**() const noexcept = 0

virtual void **setRole**(const char *role) noexcept = 0

If role is a valid role, other parameters are ignored and menu will contain only that role.

virtual const char ***getRole**() const noexcept = 0

virtual void **setIncludeColorSpaces**(bool include) noexcept = 0

Include all color spaces (or not) to ColorSpaceMenuHelper. Default is to include color spaces.

virtual bool **getIncludeColorSpaces**() const noexcept = 0

virtual *SearchReferenceSpaceType* **getSearchReferenceSpaceType**() const noexcept = 0

Can be used to restrict the search using the ReferenceSpaceType of the color spaces. It has no effect on roles and named transforms.

virtual void **setSearchReferenceSpaceType**(*SearchReferenceSpaceType* colorSpaceType) noexcept = 0

virtual void **setIncludeNamedTransforms**(bool include) noexcept = 0

Include all named transforms (or not) to ColorSpaceMenuHelper. Default is not to include named transforms.

virtual bool **getIncludeNamedTransforms**() const noexcept = 0

virtual void **setTreatNoCategoryAsAny**(bool value) noexcept = 0

When searching for color spaces using app or user categories, treat color spaces or named transforms that have no categories as if they had any of the categories. Default is true.

virtual bool **getTreatNoCategoryAsAny**() const noexcept = 0

virtual void **setAppCategories**(const char *appCategories) noexcept = 0

App categories is a comma separated list of categories. If appCategories is not NULL and not empty, all color spaces that have one of the categories will be part of the menu.

virtual const char ***getAppCategories**() const noexcept = 0

virtual void **setEncodings**(const char *encodings) noexcept = 0

Encodings is a comma separated list of encodings. When not empty, is retricting the search to color spaces that are using one of the encodings.

virtual const char ***getEncodings**() const noexcept = 0

virtual void **setUserCategories**(const char *userCategories) noexcept = 0

User categories is a comma separated list of categories. If OCIO_USER_CATEGORIES_ENVVAR env. variable is defined and not empty, this parameter is ignored and the value of the env. variable is used for user categories.

virtual const char ***getUserCategories**() const noexcept = 0

virtual void **setIncludeRoles**(bool include) noexcept = 0

Include all roles (or not) to ColorSpaceMenuHelper. Default is not to include roles. Roles are added after color spaces with an single hierarchy level named "Roles".

virtual bool **getIncludeRoles**() const noexcept = 0

virtual void **addColorSpace**(const char *name) noexcept = 0

Add an additional color space (or named transform) to the menu.

Note that an additional color space could be:

- an inactive color space,
- an active color space not having at least one of the selected categories,
- a newly created color space. Will throw when creating the menu if color space is not part of the config. Nothing is done if it is already part of the menu. It's ok to call this multiple times with the same color space, it will only be added to the menu once. If a role name is passed in, the name in the menu will be the color space name the role points to.

virtual size_t **getNumAddedColorSpaces**() const noexcept = 0

virtual const char ***getAddedColorSpace**(size_t index) const noexcept = 0

virtual void **clearAddedColorSpaces**() noexcept = 0

virtual **~ColorSpaceMenuParameters**() = default

Do not use (needed only for pybind11).

Public Static Functions

static ColorSpaceMenuParametersRcPtr **Create**(*ConstConfigRcPtr* config)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ColorSpaceMenuParameters*&)

class **ColorSpaceMenuHelper**

Helper class to create menus for the content of a config.

Menu can list color spaces, roles, named transforms. Each entry has a name, a UI name, a description, and a family. Family can also be accessed as hierarchy levels; levels are created by splitting the family using the 'family separator'. Hierarchy levels are meant to be used as sub-menus.

The UI name is what is intended to be put in application menus seen by the end-user. However, please note that the UI name is not guaranteed to remain stable between releases and so if applications need to save something it should be the 'name' rather than the 'UI name'. Currently, the only difference between the 'name' and 'UI name' is for roles.

The overall ordering of items is: color spaces, named transforms, roles, and additional color spaces. The display color spaces will either come before or after the other color spaces based on where that block of spaces appears

in the config. The order of items returned by the menu helper preserves the order of items in the config itself for each type of elements, thus preserving the intent of the config author. For example, if you call getName at idx and idx+1, the name returned at idx+1 will be from farther down in the config than the one at idx as long as both are of the same type. (An application may ask for only the items in one of those blocks if it wants to handle them separately.) If the application makes use of hierarchical menus, that will obviously impose a different order on what the user sees in the menu. Though even with hierarchical menus, applications should try to preserve config ordering (which is equivalent to index ordering) for items within the same sub-menu.

Public Functions

virtual size_t **getNumColorSpaces**() const noexcept = 0

Access to the color spaces (or roles).

virtual const char ***getName**(size_t idx) const noexcept = 0

Get the color space (or role) name used in the config for this menu item. Will be empty if the index is out of range.

virtual const char ***getUIName**(size_t idx) const noexcept = 0

Get the name to use in the menu UI. This might be different from the config name, for example in the case of roles. Will be empty if the index is out of range.

virtual size_t **getIndexFromName**(const char *name) const noexcept = 0

Get the index of the element of a given name. Return (size_t)-1 name if NULL or empty, or if no element with that name is found.

virtual size_t **getIndexFromUIName**(const char *name) const noexcept = 0

virtual const char ***getDescription**(size_t idx) const noexcept = 0

virtual const char ***getFamily**(size_t idx) const noexcept = 0

virtual size_t **getNumHierarchyLevels**(size_t idx) const noexcept = 0

Hierarchy levels are created from the family string. It is split into levels using the 'family separator'.

virtual const char ***getHierarchyLevel**(size_t idx, size_t i) const noexcept = 0

virtual const char ***getNameFromUIName**(const char *uiName) const noexcept = 0

Get the color space name from the UI name.

virtual const char ***getUINameFromName**(const char *name) const noexcept = 0

Get the color space UI name from the name.

ColorSpaceMenuHelper(const *ColorSpaceMenuHelper*&) = delete

ColorSpaceMenuHelper &**operator=**(const *ColorSpaceMenuHelper*&) = delete

virtual **~ColorSpaceMenuHelper**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *ColorSpaceMenuHelperRcPtr* **Create**(ConstColorSpaceMenuParametersRcPtr parameters)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ColorSpaceMenuHelper*&)

typedef std::shared_ptr<const *ColorSpaceMenuHelper*>
OpenColorIO_v2_5::**ConstColorSpaceMenuHelperRcPtr**

typedef std::shared_ptr<*ColorSpaceMenuHelper*> OpenColorIO_v2_5::**ColorSpaceMenuHelperRcPtr**

DisplayViewHelpers

Python

PyOpenColorIO.DisplayViewHelpers.**GetProcessor**(*config: PyOpenColorIO.PyOpenColorIO.Config*, *context: PyOpenColorIO.PyOpenColorIO.Context = None*, *workingSpaceName: str*, *displayName: str*, *viewName: str*, *channelView: PyOpenColorIO.PyOpenColorIO.MatrixTransform = None*, *direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>*) → *PyOpenColorIO.PyOpenColorIO.Processor*

Get the processor from the working color space to (display, view) pair (forward) or (display, view) pair to working (inverse). The working color space name could be a role name or a color space name. ChannelView can be empty. If not already present, each of these functions adds ExposureContrastTransforms to enable changing exposure, contrast, and gamma after the processor has been created using dynamic properties.

PyOpenColorIO.DisplayViewHelpers.**GetIdentityProcessor**(*config:* PyOpenColorIO.PyOpenColorIO.Config) → *PyOpenColorIO.PyOpenColorIO.Processor*

Get an identity processor containing only the ExposureContrastTransforms.

PyOpenColorIO.DisplayViewHelpers.**AddDisplayView**(*config: PyOpenColorIO.PyOpenColorIO.Config*, *displayName: str*, *viewName: str*, *lookName: str = ''*, *colorSpaceName: str = ''*, *colorSpaceFamily: str = ''*, *colorSpaceDescription: str = ''*, *colorSpaceCategories: str = ''*, *transformFilePath: str*, *connectionColorSpaceName: str*) → None

Add a new (display, view) pair and the new color space to a configuration instance. The input to the userTransform must be in the specified connectionColorSpace.

PyOpenColorIO.DisplayViewHelpers.**RemoveDisplayView**(*config:* PyOpenColorIO.PyOpenColorIO.Config, *displayName: str*, *viewName: str*) → None

Remove a (display, view) pair including the associated color space (only if not used). Note that the view is always removed but the display is only removed if empty.

C++

namespace **DisplayViewHelpers**

Functions

ConstProcessorRcPtr **GetProcessor**(const *ConstConfigRcPtr* &config, const *ConstContextRcPtr* &context, const char *workingName, const char *displayName, const char *viewName, const *ConstMatrixTransformRcPtr* &channelView, *TransformDirection* direction)

Get the processor from the working color space to (display, view) pair (forward) or (display, view) pair to working (inverse). The working color space name could be a role name or a color space name. ChannelView can be empty. If not already present, each of these functions adds ExposureContrastTransforms to enable changing exposure, contrast, and gamma after the processor has been created using dynamic properties.

ConstProcessorRcPtr **GetProcessor**(const *ConstConfigRcPtr* &config, const char *workingName, const char *displayName, const char *viewName, const *ConstMatrixTransformRcPtr* &channelView, *TransformDirection* direction)

ConstProcessorRcPtr **GetIdentityProcessor**(const *ConstConfigRcPtr* &config)

Get an identity processor containing only the ExposureContrastTransforms.

void **AddDisplayView**(*ConfigRcPtr* &config, const char *displayName, const char *viewName, const char *lookDefinition, const char *colorSpaceName, const char *colorSpaceFamily, const char *colorSpaceDescription, const char *categories, const char *transformFilePath, const char *connectionColorSpaceName)

Add a new (display, view) pair and the new color space to a configuration instance. The input to the userTransform must be in the specified connectionColorSpace.

void **RemoveDisplayView**(*ConfigRcPtr* &config, const char *displayName, const char *viewName)

Remove a (display, view) pair including the associated color space (only if not used). Note that the view is always removed but the display is only removed if empty.

LegacyViewingPipeline

Python

class PyOpenColorIO.**LegacyViewingPipeline**

Whereas the DisplayViewTransform simply applies a specific view from an OCIO display, the LegacyViewingPipeline provides an example of a complete viewing pipeline of the sort that could be used to implement a viewport in a typical application. It therefore adds, around the DisplayViewTransform, various optional color correction steps and RGBA channel view swizzling. The direction of the DisplayViewTranform is used as the direction of the pipeline. Note: The LegacyViewingPipeline class provides the same functionality as the OCIO v1 DisplayTransform.

Legacy viewing pipeline: - Start in display transform input color space. - If linearCC is provided: - Go to scene_linear colorspace. - Apply linearCC transform.

- If colorTimingCC is provided:
- Go to color_timing colorspace.
- Apply colorTimingCC transform.
- Apply looks (from display transform or from looks override).
- Go to first look color space.
- Apply first look transform.
- Iterate for all looks.

- Apply channelView transform.
- Apply display transform (without looks).
- Apply displayCC. Note that looks are applied even if the display transform involves data color spaces.

LegacyViewingPipeline → None

getChannelView() → *PyOpenColorIO.PyOpenColorIO.Transform*

getColorTimingCC() → *PyOpenColorIO.PyOpenColorIO.Transform*

getDisplayCC() → *PyOpenColorIO.PyOpenColorIO.Transform*

getDisplayViewTransform() → *PyOpenColorIO.PyOpenColorIO.DisplayViewTransform*

getLinearCC() → *PyOpenColorIO.PyOpenColorIO.Transform*

getLooksOverride() → str

getLooksOverrideEnabled() → bool

getProcessor(*config:* PyOpenColorIO.PyOpenColorIO.Config, *context:* PyOpenColorIO.PyOpenColorIO.Context = *None*) → *PyOpenColorIO.PyOpenColorIO.Processor*

setChannelView(*arg0:* PyOpenColorIO.PyOpenColorIO.Transform) → None

setColorTimingCC(*arg0:* PyOpenColorIO.PyOpenColorIO.Transform) → None

setDisplayCC(*arg0:* PyOpenColorIO.PyOpenColorIO.Transform) → None

setDisplayViewTransform(*arg0:* PyOpenColorIO.PyOpenColorIO.DisplayViewTransform) → None

setLinearCC(*arg0:* PyOpenColorIO.PyOpenColorIO.Transform) → None

setLooksOverride(*looks: str*) → None

A user can optionally override the looks that are, by default, used with the expected display / view combination. A common use case for this functionality is in an image viewing app, where per-shot looks are supported. If for some reason a per-shot look is not defined for the current Context, the Config::getProcessor fcn will not succeed by default. Thus, with this mechanism the viewing app could override to looks = "", and this will allow image display to continue (though hopefully) the interface would reflect this fallback option.

Looks is a potentially comma (or colon) delimited list of lookNames, where +/- prefixes are optionally allowed to denote forward/inverse look specification (and forward is assumed in the absence of either).

setLooksOverrideEnabled(*arg0: bool*) → None

Specify whether the lookOverride should be used, or not. This is a separate flag, as it's often useful to override "looks" to an empty string.

C++

class **LegacyViewingPipeline**

Whereas the DisplayViewTransform simply applies a specific view from an OCIO display, the LegacyViewingPipeline provides an example of a complete viewing pipeline of the sort that could be used to implement a viewport in a typical application. It therefore adds, around the DisplayViewTransform, various optional color correction steps and RGBA channel view swizzling. The direction of the DisplayViewTranform is used as the direction of the pipeline. Note: The LegacyViewingPipeline class provides the same functionality as the OCIO v1 DisplayTransform.

Legacy viewing pipeline:

- Start in display transform input color space.
- If linearCC is provided:
  - Go to scene_linear colorspace.
  - Apply linearCC transform.
- If colorTimingCC is provided:
  - Go to color_timing colorspace.
  - Apply colorTimingCC transform.
- Apply looks (from display transform or from looks override).
  - Go to first look color space.
  - Apply first look transform.
  - Iterate for all looks.
- Apply channelView transform.
- Apply display transform (without looks).
- Apply displayCC. Note that looks are applied even if the display transform involves data color spaces.

Public Functions

virtual *ConstDisplayViewTransformRcPtr* **getDisplayViewTransform**() const noexcept = 0

virtual void **setDisplayViewTransform**(const *ConstDisplayViewTransformRcPtr* &dt) noexcept = 0

virtual *ConstTransformRcPtr* **getLinearCC**() const noexcept = 0

virtual void **setLinearCC**(const *ConstTransformRcPtr* &cc) noexcept = 0

virtual *ConstTransformRcPtr* **getColorTimingCC**() const noexcept = 0

virtual void **setColorTimingCC**(const *ConstTransformRcPtr* &cc) noexcept = 0

virtual *ConstTransformRcPtr* **getChannelView**() const noexcept = 0

virtual void **setChannelView**(const *ConstTransformRcPtr* &transform) noexcept = 0

virtual *ConstTransformRcPtr* **getDisplayCC**() const noexcept = 0

virtual void **setDisplayCC**(const *ConstTransformRcPtr* &cc) noexcept = 0

virtual void **setLooksOverrideEnabled**(bool enable) = 0

Specify whether the lookOverride should be used, or not. This is a separate flag, as it's often useful to override "looks" to an empty string.

virtual bool **getLooksOverrideEnabled**() const = 0

virtual void **setLooksOverride**(const char *looks) = 0

A user can optionally override the looks that are, by default, used with the expected display / view combination. A common use case for this functionality is in an image viewing app, where per-shot looks are supported. If for some reason a per-shot look is not defined for the current Context, the Config::getProcessor fcn will not succeed by default. Thus, with this mechanism the viewing app could override to looks = "", and this will allow image display to continue (though hopefully) the interface would reflect this fallback option.

Looks is a potentially comma (or colon) delimited list of lookNames, where +/- prefixes are optionally allowed to denote forward/inverse look specification (and forward is assumed in the absence of either).

virtual const char ***getLooksOverride**() const = 0

virtual *ConstProcessorRcPtr* **getProcessor**(const *ConstConfigRcPtr* &config, const *ConstContextRcPtr* &context) const = 0

virtual *ConstProcessorRcPtr* **getProcessor**(const *ConstConfigRcPtr* &config) const = 0

LegacyViewingPipeline(const *LegacyViewingPipeline*&) = delete

LegacyViewingPipeline &**operator=**(const *LegacyViewingPipeline*&) = delete

virtual **~LegacyViewingPipeline**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *LegacyViewingPipelineRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *LegacyViewingPipeline*&)

typedef std::shared_ptr<const *LegacyViewingPipeline*>
OpenColorIO_v2_5::**ConstLegacyViewingPipelineRcPtr**

typedef std::shared_ptr<*LegacyViewingPipeline*> OpenColorIO_v2_5::**LegacyViewingPipelineRcPtr**

MixingHelpers

Python

class PyOpenColorIO.**MixingSlider**

The MixingSlider and MixingColorSpaceManager classes are to help applications implement correct color pickers. The term "color mixing" is used here to describe what is done in a typical application "color picker" user interface.

A user may want to mix colors in different color spaces. The two most common mixing space options are a scene-linear working space or the display space.

Since scene-linear color spaces are not perceptually uniform, it is necessary to compensate UI widgets such as sliders. For example, it is nice if mid-gray falls near the center of mixing controls rather than way over near the black end. This may be done by using a mapping from linear into an approximately perceptually uniform space.

Also note that a color picking/mixing UI may want to present a given color space in several different encodings. The most common two encodings for color mixing are RGB and HSV.

Note that these helpers anticipate that a user may want to mix colors using values that extend outside the typical [0,1] domain.

MixingSlider(**args*, ***kwargs*)

getSliderMaxEdge() → float

Maximum edge of a UI slider for conversion to mixing space.

getSliderMinEdge() → float

Minimum edge of a UI slider for conversion to mixing space.

mixingToSlider(*mixingUnits: SupportsFloat*) → float

Convert from mixing space units to distance along the slider.

setSliderMaxEdge(*arg0: SupportsFloat*) → None

Set the maximum edge of a UI slider for conversion to mixing space.

setSliderMinEdge(*arg0: SupportsFloat*) → None

Set the minimum edge of a UI slider for conversion to mixing space.

sliderToMixing(*sliderUnits: SupportsFloat*) → float

Convert from units in distance along the slider to mixing space units.

class PyOpenColorIO.**MixingColorSpaceManager**

Used to mix (or pick/choose) colors.

MixingColorSpaceManager(*config:* PyOpenColorIO.PyOpenColorIO.Config) → None

getMixingEncodings() → *PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator*

getMixingSpaces() → *PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator*

getProcessor(*workingSpaceName: str*, *displayName: str*, *viewName: str*, *direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>*) → *PyOpenColorIO.PyOpenColorIO.Processor*

getSelectedMixingEncodingIdx() → int

getSelectedMixingSpaceIdx() → int

getSlider(**args*, ***kwargs*)

Overloaded function.

1. getSlider(self: PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager) -> PyOpenColorIO.PyOpenColorIO.MixingSlider

2. getSlider(self: PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager, sliderMixingMinEdge: typing.SupportsFloat, sliderMixingMaxEdge: typing.SupportsFloat) -> PyOpenColorIO.PyOpenColorIO.MixingSlider

isPerceptuallyUniform() → bool

refresh(*config:* PyOpenColorIO.PyOpenColorIO.Config) → None

Refresh the instance (i.e. needed following a configuration change for example).

setSelectedMixingEncoding(*mixingEncoding: str*) → None

setSelectedMixingEncodingIdx(*arg0: SupportsInt*) → None

setSelectedMixingSpace(*mixingSpace: str*) → None

setSelectedMixingSpaceIdx(*arg0: SupportsInt*) → None

class PyOpenColorIO.MixingColorSpaceManager.**MixingSpaceIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.MixingColorSpaceManager.**MixingEncodingIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator*

len(*self*) → int

next(*self*) → str

C++

class **MixingSlider**

The MixingSlider and MixingColorSpaceManager classes are to help applications implement correct color pickers. The term "color mixing" is used here to describe what is done in a typical application "color picker" user interface.

A user may want to mix colors in different color spaces. The two most common mixing space options are a scene-linear working space or the display space.

Since scene-linear color spaces are not perceptually uniform, it is necessary to compensate UI widgets such as sliders. For example, it is nice if mid-gray falls near the center of mixing controls rather than way over near the black end. This may be done by using a mapping from linear into an approximately perceptually uniform space.

Also note that a color picking/mixing UI may want to present a given color space in several different encodings. The most common two encodings for color mixing are RGB and HSV.

Note that these helpers anticipate that a user may want to mix colors using values that extend outside the typical [0,1] domain.

Public Functions

virtual void **setSliderMinEdge**(float sliderMixingMinEdge) noexcept = 0

Set the minimum edge of a UI slider for conversion to mixing space.

virtual float **getSliderMinEdge**() const noexcept = 0

Minimum edge of a UI slider for conversion to mixing space.

virtual void **setSliderMaxEdge**(float sliderMixingMaxEdge) noexcept = 0

Set the maximum edge of a UI slider for conversion to mixing space.

virtual float **getSliderMaxEdge**() const noexcept = 0

Maximum edge of a UI slider for conversion to mixing space.

virtual float **sliderToMixing**(float sliderUnits) const noexcept = 0

Convert from units in distance along the slider to mixing space units.

virtual float **mixingToSlider**(float mixingUnits) const noexcept = 0

Convert from mixing space units to distance along the slider.

MixingSlider(const *MixingSlider*&) = delete

MixingSlider &**operator=**(const *MixingSlider*&) = delete

virtual **~MixingSlider**() = default

Do not use (needed only for pybind11).

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *MixingSlider*&)

class **MixingColorSpaceManager**

Used to mix (or pick/choose) colors.

Public Functions

virtual size_t **getNumMixingSpaces**() const noexcept = 0

Access to the mixing spaces.

virtual const char ***getMixingSpaceUIName**(size_t idx) const = 0

virtual size_t **getSelectedMixingSpaceIdx**() const noexcept = 0

virtual void **setSelectedMixingSpaceIdx**(size_t idx) = 0

virtual void **setSelectedMixingSpace**(const char *mixingSpace) = 0

virtual bool **isPerceptuallyUniform**() const noexcept = 0

virtual size_t **getNumMixingEncodings**() const noexcept = 0

Access to the mixing encodings.

virtual const char ***getMixingEncodingName**(size_t idx) const = 0

virtual size_t **getSelectedMixingEncodingIdx**() const noexcept = 0

virtual void **setSelectedMixingEncodingIdx**(size_t idx) = 0

virtual void **setSelectedMixingEncoding**(const char *mixingEncoding) = 0

virtual void **refresh**(*ConstConfigRcPtr* config) = 0

Refresh the instance (i.e. needed following a configuration change for example).

virtual *ConstProcessorRcPtr* **getProcessor**(const char *workingName, const char *displayName, const char *viewName, *TransformDirection* direction) const = 0

virtual *MixingSlider* &**getSlider**() noexcept = 0

virtual *MixingSlider* &**getSlider**(float sliderMixingMinEdge, float sliderMixingMaxEdge) noexcept = 0

MixingColorSpaceManager(const *MixingColorSpaceManager*&) = delete

MixingColorSpaceManager &**operator=**(const *MixingColorSpaceManager*&) = delete

virtual **~MixingColorSpaceManager**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *MixingColorSpaceManagerRcPtr* **Create**(*ConstConfigRcPtr* &config)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *MixingColorSpaceManager*&)

typedef std::shared_ptr<const *MixingColorSpaceManager*>
OpenColorIO_v2_5::**ConstMixingColorSpaceManagerRcPtr**

typedef std::shared_ptr<*MixingColorSpaceManager*> OpenColorIO_v2_5::**MixingColorSpaceManagerRcPtr**

5.7.2 Baker

Python

class PyOpenColorIO.**Baker**

In certain situations it is necessary to serialize transforms into a variety of application specific LUT formats. Note that not all file formats that may be read also support baking.

Usage Example: *Bake a CSP sRGB viewer LUT*

```
OCIO::ConstConfigRcPtr config = OCIO::Config::CreateFromEnv();
OCIO::BakerRcPtr baker = OCIO::Baker::Create();
baker->setConfig(config);
baker->setFormat("csp");
baker->setInputSpace("lnf");
baker->setShaperSpace("log");
baker->setTargetSpace("sRGB");
auto & metadata = baker->getFormatMetadata();
metadata.addChildElement(OCIO::METADATA_DESCRIPTION, "A first comment");
metadata.addChildElement(OCIO::METADATA_DESCRIPTION, "A second comment");
std::ostringstream out;
baker->bake(out); // fresh bread anyone!
std::cout << out.str();
```

Baker(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.Baker) -> None

Create a new Baker.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.Baker, config: PyOpenColorIO.PyOpenColorIO.Config, format: str, inputSpace: str, targetSpace: str, looks: str = '', cubeSize: typing.SupportsInt = -1, shaperSpace: str = '', shaperSize: typing.SupportsInt = -1) -> None

Create a new Baker.

bake(**args*, ***kwargs*)

Overloaded function.

1. bake(self: PyOpenColorIO.PyOpenColorIO.Baker, fileName: str) -> None
2. bake(self: PyOpenColorIO.PyOpenColorIO.Baker) -> str

Bake the LUT into the output stream.

getConfig() → *PyOpenColorIO.PyOpenColorIO.Config*

getCubeSize() → int

getDisplay() → str

getFormat() → str

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

static getFormats() → *PyOpenColorIO.PyOpenColorIO.Baker.FormatIterator*

getInputSpace() → str

getLooks() → str

getShaperSize() → int

getShaperSpace() → str

getTargetSpace() → str

getView() → str

setConfig(*config:* PyOpenColorIO.PyOpenColorIO.Config) → None

Set the config to use.

setCubeSize(*cubeSize: SupportsInt*) → None

Override the main LUT (3d or 1d) sample size. Default value is -1, which allows each format to use its own most appropriate size.

setDisplayView(*display: str*, *view: str*) → None

Set the display and view to apply during the baking. Must not be used if setTargetSpace is used.

setFormat(*formatName: str*) → None

Set the LUT output format.

setInputSpace(*inputSpace: str*) → None

Set the input ColorSpace that the LUT will be applied to.

setLooks(*looks: str*) → None

Set the looks to be applied during baking. Looks is a potentially comma (or colon) delimited list of lookNames, where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either).

setShaperSize(*shaperSize: SupportsInt*) → None

Override the default shaper LUT size. Default value is -1, which allows each format to use its own most appropriate size. For the CLF format, the default uses a half-domain LUT1D (which is ideal for scene-linear inputs).

setShaperSpace(*shaperSpace: str*) → None

Set an *optional* ColorSpace or NamedTransform to shape the incoming values of the LUT. When baking 3DLUT, this will correspond to the 1D shaper used to normalise incoming values to the unit range. When baking 1D LUT, this will be used to determine the input range of the LUT.

setTargetSpace(*targetSpace: str*) → None

Set the target (i.e., output) color space for the LUT. Must not be used if setDisplayView is used.

class PyOpenColorIO.Baker.**FormatIterator**

Iterator on LUT baker Formats.

Each item is a tuple containing format name and format extension.

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Baker.FormatIterator*

len(*self*) → int

next(*self*) → tuple

C++

class **Baker**

In certain situations it is necessary to serialize transforms into a variety of application specific LUT formats. Note that not all file formats that may be read also support baking.

Usage Example: *Bake a CSP sRGB viewer LUT*

```
OCIO::ConstConfigRcPtr config = OCIO::Config::CreateFromEnv();
OCIO::BakerRcPtr baker = OCIO::Baker::Create();
baker->setConfig(config);
baker->setFormat("csp");
baker->setInputSpace("lnf");
baker->setShaperSpace("log");
baker->setTargetSpace("sRGB");
auto & metadata = baker->getFormatMetadata();
metadata.addChildElement(OCIO::METADATA_DESCRIPTION, "A first comment");
metadata.addChildElement(OCIO::METADATA_DESCRIPTION, "A second comment");
std::ostringstream out;
baker->bake(out); // fresh bread anyone!
std::cout << out.str();
```

Public Functions

BakerRcPtr **createEditableCopy**() const

Create a copy of this Baker.

ConstConfigRcPtr **getConfig**() const

void **setConfig**(const *ConstConfigRcPtr* &config)

Set the config to use.

const char ***getFormat**() const

void **setFormat**(const char *formatName)

Set the LUT output format.

const *FormatMetadata* &**getFormatMetadata**() const

FormatMetadata &**getFormatMetadata**()

Get editable *optional* format metadata. The metadata that will be used varies based on the capability of the given file format. Formats such as CSP, IridasCube, and ResolveCube will create comments in the file header using the value of any first-level children elements of the formatMetadata. The CLF/CTF formats will make use of the top-level "id" and "name" attributes and children elements "Description", "InputDescriptor", "OutputDescriptor", and "Info".

const char *__getInputSpace__() const

void **setInputSpace**(const char *inputSpace)

Set the input ColorSpace that the LUT will be applied to.

const char *__getShaperSpace__() const

void **setShaperSpace**(const char *shaperSpace)

Set an *optional* ColorSpace or NamedTransform to shape the incoming values of the LUT. When baking 3DLUT, this will correspond to the 1D shaper used to normalise incoming values to the unit range. When baking 1D LUT, this will be used to determine the input range of the LUT.

const char *__getLooks__() const

void **setLooks**(const char *looks)

Set the looks to be applied during baking. Looks is a potentially comma (or colon) delimited list of lookNames, where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either).

const char *__getTargetSpace__() const

void **setTargetSpace**(const char *targetSpace)

Set the target (i.e., output) color space for the LUT. Must not be used if setDisplayView is used.

const char *__getDisplay__() const

const char *__getView__() const

void **setDisplayView**(const char *display, const char *view)

Set the display and view to apply during the baking. Must not be used if setTargetSpace is used.

int **getShaperSize**() const

void **setShaperSize**(int shapersize)

Override the default shaper LUT size. Default value is -1, which allows each format to use its own most appropriate size. For the CLF format, the default uses a half-domain LUT1D (which is ideal for scene-linear inputs).

int **getCubeSize**() const

void **setCubeSize**(int cubesize)

Override the main LUT (3d or 1d) sample size. Default value is -1, which allows each format to use its own most appropriate size.

void **bake**(std::ostream &os) const

Bake the LUT into the output stream.

Baker(const *Baker*&) = delete

Baker &**operator=**(const *Baker*&) = delete

~Baker()

Do not use (needed only for pybind11).

Public Static Functions

static *BakerRcPtr* **Create**()

Create a new Baker.

static int **getNumFormats**()

Get the number of LUT bakers.

static const char ***getFormatNameByIndex**(int index)

Get the LUT baker format name at index, return empty string if an invalid index is specified.

static const char ***getFormatExtensionByIndex**(int index)

Get the LUT baker format extension at index, return empty string if an invalid index is specified.

typedef std::shared_ptr<const *Baker*> OpenColorIO_v2_5::**ConstBakerRcPtr**

typedef std::shared_ptr<*Baker*> OpenColorIO_v2_5::**BakerRcPtr**

5.7.3 ColorSpace

Python

class PyOpenColorIO.**ColorSpace**

The *:ref: `ColorSpace `* is the state of an image with respect to colorimetry and color encoding. Transforming images between different *ColorSpaces* is the primary motivation for this library.

While a complete discussion of color spaces is beyond the scope of header documentation, traditional uses would be to have *ColorSpaces* corresponding to: physical capture devices (known cameras, scanners), and internal 'convenience' spaces (such as scene linear, logarithmic).

ColorSpace(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpace) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpace, referenceSpace: PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType) -> None
3. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpace, referenceSpace: PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType = <ReferenceSpaceType.REFERENCE_SPACE_SCENE: 0>, name: str = '', aliases: collections.abc.Sequence[str] = [], family: str = '', encoding: str = '', equalityGroup: str = '', description: str = '', bitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth = <BitDepth.BIT_DEPTH_UNKNOWN: 0>, isData: bool = False, allocation: PyOpenColorIO.PyOpenColorIO.Allocation = <Allocation.ALLOCATION_UNIFORM: 1>, allocationVars: collections.abc.Sequence[typing.SupportsFloat] = [], toReference: PyOpenColorIO.PyOpenColorIO.Transform = None, fromReference: PyOpenColorIO.PyOpenColorIO.Transform = None, categories: collections.abc.Sequence[str] = [], interopID: str = '') -> None

addAlias(*alias: str*) → None

Add an alias for the color space name (the aliases may be used as a synonym for the name). Nothing will be added if the alias is already the color space name, one of its aliases, or the argument is null. The aliases must not conflict with existing roles, color space names, named transform names, or other aliases. This is verified when adding the color space to the config.

addCategory(*category: str*) → None

Add a single category.

Note: Will do nothing if the category already exists.

---

clearAliases() → None

clearCategories() → None

Clear all the categories.

getAliases() → *PyOpenColorIO.PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator*

getAllocation() → *PyOpenColorIO.PyOpenColorIO.Allocation*

Allocation

If this colorspace needs to be transferred to a limited dynamic range coding space (such as during display with a GPU path), use this allocation to maximize bit efficiency.

getAllocationVars() → list[float]

getBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

getCategories() → *PyOpenColorIO.PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator*

getDescription() → str

getEncoding() → str

Encodings

It is sometimes useful for applications to group color spaces based on how the color values are digitally encoded. For example, images in scene-linear, logarithmic, video, and data color spaces could have different default views. Unlike the Family and EqualityGroup attributes of a color space, the list of Encodings is predefined in the OCIO documentation (rather than being config-specific) to make it easier for applications to utilize.

Here is an example config entry that could appear under a ColorSpace:

```
encoding: scene-linear
```

Encoding strings are not case-sensitive. Although users may add their own encodings, the strings will typically come from a fixed set listed in the documentation (similar to roles).

getEqualityGroup() → str

Get the ColorSpace group name (used for equality comparisons) This allows no-op transforms between different colorspaces. If an equalityGroup is not defined (an empty string), it will be considered unique (i.e., it will not compare as equal to other ColorSpaces with an empty equality group).

getFamily() → str

Get the family, for use in user interfaces (optional) The family string could use a '/' separator to indicate levels to be used by hierarchical menus.

getInterchangeAttribute(*attrName: str*) → str

Get/Set the interchange attributes.

Currently supported attribute names are "amf_transform_ids" and "icc_profile_name". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

The ICC profile name identifies the ICC color profile associated with this color space. This can be used to link OCIO color spaces with corresponding ICC profiles for applications that need to work with both color management systems.

getInterchangeAttributes() → dict[str, str]

getInteropID() → str

Get/Set the interop ID for the color space. The interop ID is a structured string defined by the Color Interop Forum. It is intended to identify color spaces in a way that is portable across different configs, making it suitable for use in various file formats. The Color Interop Forum publishes ID strings for common color spaces. If you create your own IDs, they must be preceded by a namespace string. The setter will throw if the string does not follow certain rules (run ociocheck for a more complete validation).

getName() → str

getReferenceSpaceType() → *PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType*

A display color space will use the display-referred reference space.

getTransform(*direction:* PyOpenColorIO.PyOpenColorIO.ColorSpaceDirection) → *PyOpenColorIO.PyOpenColorIO.Transform*

:ref: `Transform`

If a transform in the specified direction has been specified, return it. Otherwise return a null ConstTransformRcPtr

hasAlias(*alias: str*) → bool

Return true if alias exists.

hasCategory(*category: str*) → bool

Return true if the category is present.

A category is used to allow applications to filter the list of color spaces they display in menus based on what that color space is used for.

Here is an example config entry that could appear under a ColorSpace:

```
categories: [ file-io, working-space, basic-3d ]
```

The example contains three categories: 'file-io', 'working-space' and 'basic-3d'.

Note: Category strings are not case-sensitive and the order is not significant.

There is no limit imposed on length or number. Although users may add their own categories, the strings will typically come from a fixed set listed in the documentation (similar to roles).

isData() → bool

Data

ColorSpaces that are data are treated a bit special. Basically, any colorspace transforms you try to apply to them are ignored. (Think of applying a gamut mapping transform to an ID pass). However, the setDataBypass method on ColorSpaceTransform and DisplayViewTransform allow applications to process data when necessary. (Think of sending mattes to an HDR monitor.)

This is traditionally used for pixel data that represents non-color pixel data, such as normals, point positions, ID information, etc.

removeAlias(*alias: str*) → None

Does nothing if alias is not present.

removeCategory(*category: str*) → None

Remove a category.

Note: Will do nothing if the category is missing.

setAllocation(*allocation:* PyOpenColorIO.PyOpenColorIO.Allocation) → None

setAllocationVars(*vars: collections.abc.Sequence[SupportsFloat]*) → None

setBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

setDescription(*description: str*) → None

setEncoding(*encoding: str*) → None

setEqualityGroup(*equalityGroup: str*) → None

setFamily(*family: str*) → None

Set the family, for use in user interfaces (optional)

setInterchangeAttribute(*attrName: str*, *attrValue: str*) → None

setInteropID(*interopID: str*) → None

setIsData(*isData: bool*) → None

setName(*name: str*) → None

If the name is already an alias, that alias is removed.

setTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform, *direction:* PyOpenColorIO.PyOpenColorIO.ColorSpaceDirection) → None

Specify the transform for the appropriate direction. Setting the transform to null will clear it.

class PyOpenColorIO.ColorSpace.**ColorSpaceCategoryIterator**

self[*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.ColorSpace.**ColorSpaceAliasIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator*

len(*self*) → int

next(*self*) → str

C++

class **ColorSpace**

The *ColorSpace* is the state of an image with respect to colorimetry and color encoding. Transforming images between different *ColorSpaces* is the primary motivation for this library.

While a complete discussion of color spaces is beyond the scope of header documentation, traditional uses would be to have *ColorSpaces* corresponding to: physical capture devices (known cameras, scanners), and internal 'convenience' spaces (such as scene linear, logarithmic).

Public Functions

ColorSpaceRcPtr **createEditableCopy**() const

const char ***getName**() const noexcept

void **setName**(const char *name) noexcept

If the name is already an alias, that alias is removed.

size_t **getNumAliases**() const noexcept

const char ***getAlias**(size_t idx) const noexcept

Return empty string if idx is out of range.

bool **hasAlias**(const char *alias) const noexcept

Return true if alias exists.

void **addAlias**(const char *alias) noexcept

Add an alias for the color space name (the aliases may be used as a synonym for the name). Nothing will be added if the alias is already the color space name, one of its aliases, or the argument is null. The aliases must not conflict with existing roles, color space names, named transform names, or other aliases. This is verified when adding the color space to the config.

void **removeAlias**(const char *alias) noexcept

Does nothing if alias is not present.

void **clearAliases**() noexcept

const char ***getFamily**() const noexcept

Get the family, for use in user interfaces (optional) The family string could use a '/' separator to indicate levels to be used by hierarchical menus.

void **setFamily**(const char *family)

Set the family, for use in user interfaces (optional)

const char ***getEqualityGroup**() const noexcept

Get the ColorSpace group name (used for equality comparisons) This allows no-op transforms between different colorspaces. If an equalityGroup is not defined (an empty string), it will be considered unique (i.e., it will not compare as equal to other ColorSpaces with an empty equality group).

void **setEqualityGroup**(const char *equalityGroup)

const char ***getDescription**() const noexcept

void **setDescription**(const char *description)

const char ***getInteropID**() const noexcept

Get/Set the interop ID for the color space. The interop ID is a structured string defined by the Color Interop Forum. It is intended to identify color spaces in a way that is portable across different configs, making it suitable for use in various file formats. The Color Interop Forum publishes ID strings for common color spaces. If you create your own IDs, they must be preceded by a namespace string. The setter will throw if the string does not follow certain rules (run ociocheck for a more complete validation).

void **setInteropID**(const char *interopID)

const char ***getInterchangeAttribute**(const char *attrName) const

Get/Set the interchange attributes.

Currently supported attribute names are "amf_transform_ids" and "icc_profile_name". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

The ICC profile name identifies the ICC color profile associated with this color space. This can be used to link OCIO color spaces with corresponding ICC profiles for applications that need to work with both color management systems.

void **setInterchangeAttribute**(const char *attrName, const char *value)

std::map<std::string, std::string> **getInterchangeAttributes**() const noexcept

BitDepth **getBitDepth**() const noexcept

void **setBitDepth**(*BitDepth* bitDepth)

ReferenceSpaceType **getReferenceSpaceType**() const noexcept

A display color space will use the display-referred reference space.

bool **hasCategory**(const char *category) const

Return true if the category is present.

A category is used to allow applications to filter the list of color spaces they display in menus based on what that color space is used for.

Here is an example config entry that could appear under a ColorSpace:

```
categories: [ file-io, working-space, basic-3d ]
```

The example contains three categories: 'file-io', 'working-space' and 'basic-3d'.

There is no limit imposed on length or number. Although users may add their own categories, the strings will typically come from a fixed set listed in the documentation (similar to roles).

Note: Category strings are not case-sensitive and the order is not significant.

---

void **addCategory**(const char *category)

Add a single category.

Note: Will do nothing if the category already exists.

void **removeCategory**(const char *category)

Remove a category.

Note: Will do nothing if the category is missing.

int **getNumCategories**() const

Get the number of categories.

const char ***getCategory**(int index) const

Return the category name using its index.

Note: Will be null if the index is invalid.

void **clearCategories**()

Clear all the categories.

const char ***getEncoding**() const noexcept

Encodings

It is sometimes useful for applications to group color spaces based on how the color values are digitally encoded. For example, images in scene-linear, logarithmic, video, and data color spaces could have different default views. Unlike the Family and EqualityGroup attributes of a color space, the list of Encodings is predefined in the OCIO documentation (rather than being config-specific) to make it easier for applications to utilize.

Here is an example config entry that could appear under a ColorSpace:

```
encoding: scene-linear
```

Encoding strings are not case-sensitive. Although users may add their own encodings, the strings will typically come from a fixed set listed in the documentation (similar to roles).

void **setEncoding**(const char *encoding)

bool **isData**() const noexcept

Data

ColorSpaces that are data are treated a bit special. Basically, any colorspace transforms you try to apply to them are ignored. (Think of applying a gamut mapping transform to an ID pass). However, the setDataBypass method on ColorSpaceTransform and DisplayViewTransform allow applications to process data when necessary. (Think of sending mattes to an HDR monitor.)

This is traditionally used for pixel data that represents non-color pixel data, such as normals, point positions, ID information, etc.

void **setIsData**(bool isData) noexcept

Allocation **getAllocation**() const noexcept

Allocation

If this colorspace needs to be transferred to a limited dynamic range coding space (such as during display with a GPU path), use this allocation to maximize bit efficiency.

void **setAllocation**(*Allocation* allocation) noexcept

int **getAllocationNumVars**() const

Specify the optional variable values to configure the allocation. If no variables are specified, the defaults are used.

ALLOCATION_UNIFORM::

2 vars: [min, max]

ALLOCATION_LG2::

2 vars: [lg2min, lg2max] 3 vars: [lg2min, lg2max, linear_offset]

void **getAllocationVars**(float *vars) const

void **setAllocationVars**(int numvars, const float *vars)

ConstTransformRcPtr **getTransform**(*ColorSpaceDirection* dir) const noexcept

Transform

If a transform in the specified direction has been specified, return it. Otherwise return a null ConstTransformRcPtr

void **setTransform**(const *ConstTransformRcPtr* &transform, *ColorSpaceDirection* dir)

Specify the transform for the appropriate direction. Setting the transform to null will clear it.

ColorSpace(const *ColorSpace*&) = delete

ColorSpace &**operator=**(const *ColorSpace*&) = delete

~ColorSpace()

Do not use (needed only for pybind11).

Public Static Functions

static *ColorSpaceRcPtr* **Create**()

static *ColorSpaceRcPtr* **Create**(*ReferenceSpaceType* referenceSpace)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ColorSpace*&)

typedef std::shared_ptr<const *ColorSpace*> OpenColorIO_v2_5::**ConstColorSpaceRcPtr**

typedef std::shared_ptr<*ColorSpace*> OpenColorIO_v2_5::**ColorSpaceRcPtr**

5.7.4 ColorSpaceSet

Python

class PyOpenColorIO.**ColorSpaceSet**

The *:ref:`ColorSpaceSet`* is a set of color spaces (i.e. no color space duplication) which could be the result of Config::getColorSpaces or built from scratch.

Note: The color spaces are decoupled from the config ones, i.e., any changes to the set itself or to its color spaces do not affect the original color spaces from the configuration. If needed, use Config::addColorSpace to update the configuration.

---

self **&** *arg0: PyOpenColorIO.PyOpenColorIO.ColorSpaceSet* → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet*

Perform the intersection of two sets.

Note: This function provides operations on two color space sets where the result contains copied color spaces and no duplicates.

Parameters

- **lcss** –
- **rcss** –

self **==** *arg0: PyOpenColorIO.PyOpenColorIO.ColorSpaceSet* → bool

Return true if the two sets are equal.

Note: The comparison is done on the color space names (not a deep comparison).

ColorSpaceSet → None

Create an empty set of color spaces.

self **!=** *arg0: PyOpenColorIO.PyOpenColorIO.ColorSpaceSet* → bool

Return true if the two sets are different.

self **|** *arg0: PyOpenColorIO.PyOpenColorIO.ColorSpaceSet* → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet*

Perform the union of two sets.

ColorSpaceSetOperators .. note::
This function provides operations on two color space sets where the result contains copied color spaces and no duplicates.

Parameters

- **lcss** –
- **rcss** –

self – *arg0: PyOpenColorIO.PyOpenColorIO.ColorSpaceSet* → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet*

Perform the difference of two sets.

Note: This function provides operations on two color space sets where the result contains copied color spaces and no duplicates.

Parameters

- **lcss** –
- **rcss** –

addColorSpace(*colorSpace:* PyOpenColorIO.PyOpenColorIO.ColorSpace) → None

Add color space(s).

Note: If another color space is already registered with the same name, this will overwrite it. This stores a copy of the specified color space(s). Throws if one of the aliases is already assigned as a name or alias to an existing color space.

---

addColorSpaces(*colorSpaces:* PyOpenColorIO.PyOpenColorIO.ColorSpaceSet) → None

clearColorSpaces() → None

Clear all color spaces.

getColorSpace(*name: str*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

Note: Only accepts color space names (i.e. no role name).

Will return null if the name is not found.

getColorSpaceNames() → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator*

getColorSpaces() → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator*

hasColorSpace(*name: str*) → bool

Note: Only accepts color space names (i.e. no role name)

Parameters
name –

Returns
true

Returns
false

removeColorSpace(*colorSpace: str*) → None

Remove color space(s) using color space names (i.e. no role name).

Note: The removal of a missing color space does nothing.

removeColorSpaces(*colorSpaces:* PyOpenColorIO.PyOpenColorIO.ColorSpaceSet) → None

class PyOpenColorIO.ColorSpaceSet.**ColorSpaceNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.ColorSpaceSet.**ColorSpaceIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

C++

class **ColorSpaceSet**

The *ColorSpaceSet* is a set of color spaces (i.e. no color space duplication) which could be the result of Config::getColorSpaces or built from scratch.

Note: The color spaces are decoupled from the config ones, i.e., any changes to the set itself or to its color spaces do not affect the original color spaces from the configuration. If needed, use Config::addColorSpace to update the configuration.

Public Functions

ColorSpaceSetRcPtr **createEditableCopy**() const

Create a set containing a copy of all the color spaces.

bool **operator==**(const *ColorSpaceSet* &css) const

Return true if the two sets are equal.

Note: The comparison is done on the color space names (not a deep comparison).

bool **operator!=**(const *ColorSpaceSet* &css) const

Return true if the two sets are different.

int **getNumColorSpaces**() const

Return the number of color spaces.

const char *getColorSpaceNameByIndex(int index) const

Return the color space name using its index. This will be null if an invalid index is specified.

ConstColorSpaceRcPtr getColorSpaceByIndex(int index) const

Return the color space using its index. This will be empty if an invalid index is specified.

ConstColorSpaceRcPtr getColorSpace(const char *name) const

Will return null if the name is not found.

Note: Only accepts color space names (i.e. no role name).

int getColorSpaceIndex(const char *name) const

Will return -1 if the name is not found.

Note: Only accepts color space names (i.e. no role name).

bool hasColorSpace(const char *name) const

Note: Only accepts color space names (i.e. no role name)

Parameters
name –

Returns
true

Returns
false

void addColorSpace(const *ConstColorSpaceRcPtr* &cs)

Add color space(s).

Note: If another color space is already registered with the same name, this will overwrite it. This stores a copy of the specified color space(s). Throws if one of the aliases is already assigned as a name or alias to an existing color space.

void addColorSpaces(const *ConstColorSpaceSetRcPtr* &cs)

void removeColorSpace(const char *name)

Remove color space(s) using color space names (i.e. no role name).

Note: The removal of a missing color space does nothing.

void removeColorSpaces(const *ConstColorSpaceSetRcPtr* &cs)

void clearColorSpaces()

Clear all color spaces.

~ColorSpaceSet()

Do not use (needed only for pybind11).

Public Static Functions

static *ColorSpaceSetRcPtr* Create()

Create an empty set of color spaces.

Warning: doxygengroup: Cannot find group "ColorSpaceSetOperators" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

typedef std::shared_ptr<const *ColorSpaceSet*> OpenColorIO_v2_5::**ConstColorSpaceSetRcPtr**

typedef std::shared_ptr<*ColorSpaceSet*> OpenColorIO_v2_5::**ColorSpaceSetRcPtr**

5.7.5 Config

Python

class PyOpenColorIO.**Config**

A config defines all the color spaces to be available at runtime.

The color configuration (Config) is the main object for interacting with this library. It encapsulates all of the information necessary to use customized ColorSpaceTransform and DisplayViewTransform operations.

See the user-guide for more information on selecting, creating, and working with custom color configurations.

For applications interested in using only one color config at a time (this is the vast majority of apps), their API would traditionally get the global configuration and use that, as opposed to creating a new one. This simplifies the use case for plugins and bindings, as it alleviates the need to pass around configuration handles.

An example of an application where this would not be sufficient would be a multi-threaded image proxy server (daemon), which wished to handle multiple show configurations in a single process concurrently. This app would need to keep multiple configurations alive, and to manage them appropriately.

Roughly speaking, a novice user should select a default configuration that most closely approximates the use case (animation, visual effects, etc.), and set the *OCIO* environment variable to point at the root of that configuration.

Note: Initialization using environment variables is typically preferable in a multi-app ecosystem, as it allows all applications to be consistently configured.

See developers-usageexamples

class **ActiveDisplaysListIterator**

ActiveDisplaysListIterator(**args*, ***kwargs*)

class **ActiveViewsListIterator**

ActiveViewsListIterator(**args*, ***kwargs*)

static AreViewsEqual(*first:* PyOpenColorIO.PyOpenColorIO.Config, *second:* PyOpenColorIO.PyOpenColorIO.Config, *dispName: str*, *viewName: str*) → bool

Compare views in a pair of configs.

Will return false if either of the views does not exist. This will return true even if the view is display-defined in one config and a reference to a shared view in the other config (both within the same display), as long as the contents match. The description text (if any) is ignored, since it is not a functional difference.

Note that the comparison is only on the strings contained in the view definition, the function does not attempt to compare that the color spaces or view transforms being referenced are identical (only that they have the same name).

static AreVirtualViewsEqual(*first:* PyOpenColorIO.PyOpenColorIO.Config, *second:* PyOpenColorIO.PyOpenColorIO.Config, *viewName: str*) → bool

Compare virtual views in a pair of configs.

Will return false if either of the virtual views does not exist. This will return true even if the virtual view is display-defined in one config and a reference to a shared virtual view in the other config, as long as the contents match.

The description text (if any) is ignored, since it is not a functional difference.

Note that the comparison is only on the strings contained in the view definition, the function does not attempt to compare that the color spaces or view transforms being referenced are identical (only that they have the same name).

static CreateFromBuiltinConfig(*arg0: str*) → *PyOpenColorIO.PyOpenColorIO.Config*

Create a configuration using an OCIO built-in config.

Parameters
configName – Built-in config name (with or without the "ocio://" URI prefix).

Also supports the following OCIO URI format for Built-in configs: "ocio://default" - Default Built-in config. "ocio://cg-config-latest" - Latest Built-in CG config. "ocio://studio-config-latest" - Latest Built-in Studio config. "ocio://<CONFIG NAME>" - A specific Built-in config.

The available configNames are:

ACES Studio config, contains a more complete collection of color spaces and displays: "studio-config-v1.0.0_aces-v1.3_ocio-v2.1" "studio-config-v2.1.0_aces-v1.3_ocio-v2.3" "studio-config-v2.2.0_aces-v1.3_ocio-v2.4" "studio-config-v4.0.0_aces-v2.0_ocio-v2.5"

ACES CG config, basic color spaces for computer graphics apps: "cg-config-v1.0.0_aces-v1.3_ocio-v2.1" "cg-config-v2.1.0_aces-v1.3_ocio-v2.3" "cg-config-v2.2.0_aces-v1.3_ocio-v2.4" "cg-config-v4.0.0_aces-v2.0_ocio-v2.5"

More information is available at: https://github.com/AcademySoftwareFoundation/OpenColorIO-Config-ACES

Information about the available configs is available from the BuiltinConfigRegistry.

:exception Exception: If the configName is not recognized.

Returns
One of the configs built into the OCIO library.

static CreateFromConfigIOProxy(*arg0: OpenColorIO_v2_5::ConfigIOProxy*) → *PyOpenColorIO.PyOpenColorIO.Config*

Create a config from the supplied ConfigIOProxy object. This allows the calling program to directly provide the config and associated LUTs rather than reading them from the standard file system.

See the ConfigIOProxy class documentation for more info.

Parameters
ciop – ConfigIOProxy object providing access to the config's files.

:exception Exception: If the config may not be read from the proxy, or does not parse.

Returns
The Config object.

static CreateFromEnv() → *PyOpenColorIO.PyOpenColorIO.Config*

Create a configuration using the OCIO environment variable.

Also supports the OCIO URI format for Built-in configs and supports archived configs. See Config::CreateFromFile.

If the variable is missing or empty, returns the same result as Config::CreateRaw. :return: The Config object.

static CreateFromFile(*fileName: str*) → *PyOpenColorIO.PyOpenColorIO.Config*

Create a configuration using a specific config file.

Supports the OCIO URI format for Built-in configs. See Config::CreateFromBuiltinConfig.

Supports archived configs (.ocioz files).

:exception Exception: If the file may not be read or does not parse.

Returns
The Config object.

static CreateFromStream(*str: str*) → *PyOpenColorIO.PyOpenColorIO.Config*

Create a configuration using a stream.

Note that CreateFromStream does not set the working directory so the caller would need to set that separately in order to resolve FileTransforms. This function is typically only used for self-contained configs (no LUTs).

Configs created from CreateFromStream can not be archived unless the working directory is set and contains any necessary LUT files.

Parameters
istream – Stream to the config.

:exception Exception: If the stream does not parse.

Returns
The Config object.

static CreateRaw() → *PyOpenColorIO.PyOpenColorIO.Config*

Create a fall-back config.

This may be useful to allow client apps to launch in cases when the supplied config path is not loadable. :return: The Config object.

class DisplayAllIterator

DisplayAllIterator**args*, ***kwargs*

static GetProcessorFromBuiltinColorSpace(*builtinColorSpaceName: str*, *srcConfig: PyOpenColorIO.PyOpenColorIO.Config*, *srcColorSpaceName: str*) → *PyOpenColorIO.PyOpenColorIO.Processor*

See description of GetProcessorToBuiltinColorSpace.

Parameters

- **builtinColorSpaceName** – The name of the color space in the current default Built-in config.
- **srcConfig** – The user's source config.
- **srcColorSpaceName** – The name of the color space in the source config.

static GetProcessorFromConfigs(**args*, ***kwargs*)

Overloaded function.

1. GetProcessorFromConfigs(srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstColorSpaceName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

Get a processor to convert between color spaces in two separate configs.

This relies on both configs having the aces_interchange role (when srcName is scene-referred) or the role cie_xyz_d65_interchange (when srcName is display-referred) defined. An exception is thrown if that is not the case.

2. GetProcessorFromConfigs(srcContext: PyOpenColorIO.PyOpenColorIO.Context, srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, dstContext: PyOpenColorIO.PyOpenColorIO.Context, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstColorSpaceName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

3. GetProcessorFromConfigs(srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, srcInterchangeName: str, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstColorSpaceName: str, dstInterchangeName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

The srcInterchangeName and dstInterchangeName must refer to a pair of color spaces in the two configs that are the same. A role name may also be used.

Note: For all of the two-config GetProcessor functions, if either the source or destination color spaces are data spaces, the entire processor will be a no-op.

4. GetProcessorFromConfigs(srcContext: PyOpenColorIO.PyOpenColorIO.Context, srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, srcInterchangeName: str, dstContext: PyOpenColorIO.PyOpenColorIO.Context, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstColorSpaceName: str, dstInterchangeName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

5. GetProcessorFromConfigs(srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstDisplay: str, dstView: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

Get a processor to convert from a color space to a display and view in two separate configs.

6. GetProcessorFromConfigs(srcContext: PyOpenColorIO.PyOpenColorIO.Context, srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, dstContext: PyOpenColorIO.PyOpenColorIO.Context, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstView: str, dstDisplay: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

7. GetProcessorFromConfigs(srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, srcInterchangeName: str, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstDisplay: str, dstView: str, dstInterchangeName: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

The srcInterchangeName and dstInterchangeName must refer to a pair of color spaces in the two configs that are the same. A role name may also be used.

8. GetProcessorFromConfigs(srcContext: PyOpenColorIO.PyOpenColorIO.Context, srcConfig: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, srcInterchangeName: str, dstContext: PyOpenColorIO.PyOpenColorIO.Context, dstConfig: PyOpenColorIO.PyOpenColorIO.Config, dstDisplay: str, dstView: str, dstInterchangeName: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

static GetProcessorToBuiltinColorSpace(*srcConfig:* PyOpenColorIO.PyOpenColorIO.Config, *srcColorSpaceName: str*, *builtinColorSpaceName: str*) → *PyOpenColorIO.PyOpenColorIO.Processor*

Get a Processor to or from a known external color space.

These methods provide a way to interface color spaces in a config with known standard external color spaces. The set of external color space are those contained in the current default Built-in config. This includes common spaces such as "Linear Rec.709 (sRGB)", "sRGB - Texture", "ACEScg", and "ACES2065-1".

If the source config defines the necessary Interchange Role (typically "aces_interchange"), then the conversion will be well-defined and equivalent to calling GetProcessorFromConfigs with the source config and the Built-in config.

However, if the Interchange Roles are not present, heuristics will be used to try and identify a common color space in the source config that may be used to allow the conversion to proceed. If the heuristics fail to find a suitable space, an exception is thrown. The heuristics may evolve, so the results returned by this function for a given source config and color space may change in future releases of the library. However, the Interchange Roles are required in config versions 2.2 and higher, so it is hoped that the need for the heuristics will decrease over time.

Parameters

- **srcConfig** – The user's source config.
- **srcColorSpaceName** – The name of the color space in the source config.
- **builtinColorSpaceName** – The name of the color space in the current default Built-in config.

:exception Exception: if either the src or builtin interchange space cannot be identified.

static IdentifyBuiltinColorSpace(*srcConfig:* PyOpenColorIO.PyOpenColorIO.Config, *builtinConfig:* PyOpenColorIO.PyOpenColorIO.Config, *builtinColorSpaceName: str*) → str

Find the name of the color space in the source config that is the same as a color space in the default built-in config. For example, setting the builtinColorSpaceName to "sRGB - Texture" (a color space name from that config), would return the name for the corresponding sRGB texture space in the current config (or empty if it was not found). Note that this method relies on heuristics which may evolve over time and which may not work on all configs.

The method only looks at active color spaces. If the interchange roles are missing and heuristics are used, only scene-referred color spaces are searched.

Parameters

- **srcConfig** – The config to search for the desired color space.
- **builtinConfig** – The built-in config to use. See Config::CreateFromBuiltinConfig.
- **builtinColorSpaceName** – Color space name in the built-in default config.

Returns

Matching color space name from the source config. Empty if not found.

:exception Exception: if an interchange space cannot be found or the equivalent space cannot be found.

static IdentifyInterchangeSpace(*srcConfig:* PyOpenColorIO.PyOpenColorIO.Config, *srcColorSpaceName: str*, *builtinConfig:* PyOpenColorIO.PyOpenColorIO.Config, *builtinColorSpaceName: str*) → tuple[str, str]

Identify the two names of a common color space that exists in both the given config and the provided built-in config that may be used for converting color spaces between the two configs. If both configs have the interchange role set, than the color spaces set to that role will be returned. Otherwise, heuristics will be used to try and identify a known color space in the source config. These are the same heuristics that are used for other methods such as identifyBuiltinColorSpace and GetProcessorTo/FromBuiltinColorSpace.

Using this method in connection with GetProcessorFromConfigs is more efficient if you need to call GetProcessorTo/FromBuiltinColorSpace multiple times since it is only necessary to run the heuristics once (to identify the interchange spaces).

The srcColorSpaceName and builtinColorSpace name are used to decide which interchange role to use (scene- or display-referred). However, they are not used if the interchange roles are not present and the heuristics are used. It is actually only the ReferenceSpaceType of the color spaces that are used, so it is not necessary to call this function multiple times if all the spaces are of the same type. (These are the same arguments that would also be set if you were instead calling GetProcessorTo/FromBuiltinColorSpace.)

Parameters

- **srcInterchangeName** – Color space name from the source config.
- **builtinInterchangeName** – Corresponding color space name from the built-in config.
- **srcConfig** – The config to search for the desired color space.
- **srcColorSpaceName** – Color space name in the given config to convert to/from.
- **builtinConfig** – The built-in config to use. See Config::CreateFromBuiltinConfig.
- **builtinColorSpaceName** – Color space name in the default built-in config.

:exception Exception: if either the srcInterchange or builtinInterchange cannot be identified.

class ViewForViewTypeIterator

ViewForViewTypeIterator(**args*, ***kwargs*)

class VirtualViewIterator

VirtualViewIterator(**args*, ***kwargs*)

VirtualViewIterator → None

Create an empty config of the current version.

Note that an empty config will not pass validation since required elements will be missing. :return: The Config object.

str(*self*) → str

addActiveDisplay(*display: str*) → None

Add a single active display to the end of the list. If the display is already present, no action is taken.

addActiveView(*view: str*) → None

Add a single active view to the end of the list. If the view is already present, no action is taken.

addColorSpace(*colorSpace:* PyOpenColorIO.PyOpenColorIO.ColorSpace) → None

Add a color space to the configuration.

Note: If another color space is already present with the same name, this will overwrite it. This stores a copy of the specified color space.

Note: Adding a color space to a Config does not affect any ColorSpaceSet sets that have already been created.

---

addDisplaySharedView(*display: str*, *view: str*) → None

Add a (reference to a) shared view to a display.

The shared view must be part of the config. See Config::addSharedView

This will throw if: - Display or view are null or empty. - Display already has a view (shared or not) with the same name.

addDisplayView(**args*, ***kwargs*)

Overloaded function.

1. addDisplayView(self: PyOpenColorIO.PyOpenColorIO.Config, display: str, view: str, colorSpaceName: str, looks: str = '') -> None

For the (display, view) pair, specify which color space and look to use. If a look is not desired, then just pass a null or empty string.

2. addDisplayView(self: PyOpenColorIO.PyOpenColorIO.Config, display: str, view: str, viewTransform: str, displayColorSpaceName: str, looks: str = '', ruleName: str = '', description: str = '') -> None

For the (display, view) pair, specify which color space and look to use. If a look is not desired, then just pass a null or empty string.

addEnvironmentVar(*name: str*, *defaultValue: str*) → None

Add (or update) an environment variable with a default value. But it removes it if the default value is null.

addLook(*look:* PyOpenColorIO.PyOpenColorIO.Look) → None

addNamedTransform(*namedTransform:* PyOpenColorIO.PyOpenColorIO.NamedTransform) → None

addSearchPath(*path: str*) → None

Add a single search path to the end of the list.

Paths may be either absolute or relative. Relative paths are relative to the working directory. Forward slashes will be normalized to reverse for Windows. Environment (context) variables may be used in paths.

addSharedView(*view: str*, *viewTransformName: str*, *colorSpaceName: str*, *looks: str = ''*, *ruleName: str = ''*, *description: str = ''*) → None

Will throw if view or colorSpaceName are null or empty.

addViewTransform(*viewTransform:* PyOpenColorIO.PyOpenColorIO.ViewTransform) → None

addVirtualDisplaySharedView(*sharedView: str*) → None

addVirtualDisplayView(*view: str*, *viewTransformName: str*, *colorSpaceName: str*, *looks: str = ''*, *ruleName: str = ''*, *description: str = ''*) → None

archive(*arg0: str*) → None

Archive the config and its LUTs into the specified output stream.

The config is archived by serializing the Config object into a file named "config.ocio" and then walking through the current working directory and any sub-directories. Any files that have an extension matching a supported LUT file format are added to the archive. Any files that do not have an extension (or have some unsupported LUT extension, including .ocio), will not be added to the archive. To reiterate, it is the in-memory Config object that is archived, and not any .ocio file in the current working directory. The directory structure relative to the working directory is preserved. No files outside the working directory are archived so that if it is later expanded, no files will be created outside the working dir.

The reason the archive is created using all supported LUT file extensions rather than by trying to resolve all the FileTransforms in the Config to specific files is because of the goal to allow context variables to continue to work.

If a Config is created with CreateFromStream, CreateFromFile with an OCIOZ archive, or CreateFromConfigIOProxy, it cannot be archived unless the working directory is manually set to a directory that contains any necessary LUT files.

The provided output stream must be closed by the caller, if necessary (e.g., an ofstream).

Parameters
ostream – The output stream to write to.

clearActiveDisplays() → None

Clear the active displays list.

clearActiveViews() → None

Clear the active views list.

clearColorSpaces() → None

Remove all the color spaces from the configuration.

Note: Removing color spaces from a Config does not affect any ColorSpaceSet sets that have already been created.

---

clearDisplays() → None

Clear all the displays.

clearEnvironmentVars() → None

clearLooks() → None

clearNamedTransforms() → None

clearProcessorCache() → None

Control the caching of processors in the config instance. By default, caching is on. The flags allow turning caching off entirely or only turning it off if dynamic properties are being used by the processor.

clearSearchPaths() → None

clearSharedViews() → None

Clear all shared views. This will throw if any displays are still using the shared views.

clearViewTransforms() → None

clearVirtualDisplay() → None

Clear the virtual display.

filepathOnlyMatchesDefaultRule(*filePath: str*) → bool

Returns true if the only rule matched by filePath is the default rule. This is a convenience method for applications that want to require the user to manually choose a color space when strictParsing is true and no other rules match.

getActiveDisplays() → *PyOpenColorIO.PyOpenColorIO.Config.ActiveDisplaysListIterator*

getActiveViews() → *PyOpenColorIO.PyOpenColorIO.Config.ActiveViewsListIterator*

getCacheID(**args*, ***kwargs*)

Overloaded function.

1. getCacheID(self: PyOpenColorIO.PyOpenColorIO.Config) -> str

This will produce a hash of the all colorspace definitions, etc. All external references, such as files used in FileTransforms, etc., will be incorporated into the cacheID. While the contents of the files are not read, the file system is queried for relevant information (mtime, inode) so that the config's cacheID will change when the underlying luts are updated.

If a context is not provided, the current Context will be used.

If a null context is provided, file references will not be taken into account (this is essentially a hash of Config::serialize).

2. getCacheID(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context) -> str

This will produce a hash of the all colorspace definitions, etc. All external references, such as files used in FileTransforms, etc., will be incorporated into the cacheID. While the contents of the files are not read, the file system is queried for relevant information (mtime, inode) so that the config's cacheID will change when the underlying luts are updated.

If a context is not provided, the current Context will be used.

If a null context is provided, file references will not be taken into account (this is essentially a hash of Config::serialize).

getCanonicalName(*name: str*) → str

Accepts an alias, role name, named transform name, or color space name and returns the color space name or the named transform name.

getColorSpace(*name: str*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

Get the color space from all the color spaces (i.e. active and inactive) and return null if the name is not found.

Note: The fcn accepts either a color space name, role name, or alias. (Color space names take precedence over roles.)

---

getColorSpaceFromFilepath(*filePath: str*) → tuple

Get the color space of the first rule that matched filePath. (For v1 configs, this is equivalent to calling parseColorSpaceFromString with strictparsing set to false.)

getColorSpaceNames(**args*, ***kwargs*)

Overloaded function.

1. getColorSpaceNames(self: PyOpenColorIO.PyOpenColorIO.Config, searchReferenceType: PyOpenColorIO.PyOpenColorIO.SearchReferenceSpaceType, visibility: PyOpenColorIO.PyOpenColorIO.ColorSpaceVisibility) -> PyOpenColorIO.PyOpenColorIO.Config.ColorSpaceNameIterator

2. getColorSpaceNames(self: PyOpenColorIO.PyOpenColorIO.Config) -> PyOpenColorIO.PyOpenColorIO.Config.ActiveColorSpaceNameIterator

getColorSpaces(**args*, ***kwargs*)

Overloaded function.

1. getColorSpaces(self: PyOpenColorIO.PyOpenColorIO.Config, category: str) -> PyOpenColorIO.PyOpenColorIO.ColorSpaceSet

Get all active color spaces having a specific category in the order they appear in the config file.

Note: If the category is null or empty, the method returns all the active color spaces like Config::getNumColorSpaces and Config::getColorSpaceNameByIndex do.

Note: It's worth noticing that the method returns a copy of the selected color spaces decoupling the result from the config. Hence, any changes on the config do not affect the existing color space sets, and vice-versa.

---

2. getColorSpaces(self: PyOpenColorIO.PyOpenColorIO.Config, searchReferenceType: PyOpenColorIO.PyOpenColorIO.SearchReferenceSpaceType, visibility: PyOpenColorIO.PyOpenColorIO.ColorSpaceVisibility) -> PyOpenColorIO.PyOpenColorIO.Config.ColorSpaceIterator
3. getColorSpaces(self: PyOpenColorIO.PyOpenColorIO.Config) -> PyOpenColorIO.PyOpenColorIO.Config.ActiveColorSpaceIterator

getConfigIOProxy() → OpenColorIO_v2_5::ConfigIOProxy

getCurrentContext() → *PyOpenColorIO.PyOpenColorIO.Context*

getDefaultDisplay() → str

getDefaultLumaCoefs() → Annotated[list[float], 'FixedSize(3)']

Get the default coefficients for computing luma.

Note: There is no "1 size fits all" set of luma coefficients. (The values are typically different for each colorspace, and the application of them may be nonsensical depending on the intensity coding anyways). Thus, the 'right' answer is to make these functions on the ColorSpace class. However, it's often useful to have a config-wide default so here it is. We will add the colorspace specific luma call if/when another client is interesting in using it.

---

getDefaultSceneToDisplayViewTransform() → *PyOpenColorIO.PyOpenColorIO.ViewTransform*

This view transform is the one that will be used by default if a ColorSpaceTransform is needed between a scene-referred and display-referred color space. The config author may specify a transform to use via the default_view_transform entry in the config. If that is not present, or does not return a valid view transform from the scene-referred connection space, the fall-back is to use the first valid view transform in the config. Returns a null ConstTransformRcPtr if there isn't one.

getDefaultView(**args*, ***kwargs*)

Overloaded function.

1. getDefaultView(self: PyOpenColorIO.PyOpenColorIO.Config, display: str) -> str

2. getDefaultView(self: PyOpenColorIO.PyOpenColorIO.Config, display: str, colorSpacename: str) -> str

getDefaultViewTransformName() → str

Get or set the default_view_transform string from the config.

Note that if this is not the name of a valid view transform from the scene-referred connection space, it will be ignored.

getDescription() → str

getDisplayViewColorSpaceName(*display: str*, *view: str*) → str

Returns the colorspace attribute of the (display, view) pair. (Note that this may be either a color space or a display color space.)

getDisplayViewDescription(*display: str*, *view: str*) → str

Returns the description attribute of a (display, view) pair.

getDisplayViewLooks(*display: str*, *view: str*) → str

Returns the looks attribute of a (display, view) pair.

getDisplayViewRule(*display: str*, *view: str*) → str

Returns the rule attribute of a (display, view) pair.

getDisplayViewTransformName(*display: str*, *view: str*) → str

Returns the view_transform attribute of the (display, view) pair. View can be a shared view of the display. If display is null or empty, config shared views are used.

getDisplays() → *PyOpenColorIO.PyOpenColorIO.Config.DisplayIterator*

getDisplaysAll() → *PyOpenColorIO.PyOpenColorIO.Config.DisplayAllIterator*

getEnvironmentMode() → *PyOpenColorIO.PyOpenColorIO.EnvironmentMode*

getEnvironmentVarDefault(*name: str*) → str

getEnvironmentVarNames() → *PyOpenColorIO.PyOpenColorIO.Config.EnvironmentVarNameIterator*

getFamilySeparator() → str

Get the family separator.

A single character used to separate the family string into tokens for use in hierarchical menus. Defaults to '/'.

getFileRules() → *PyOpenColorIO.PyOpenColorIO.FileRules*

Get read-only version of the file rules.

getInactiveColorSpaces() → str

getLook(*name: str*) → *PyOpenColorIO.PyOpenColorIO.Look*

getLookNames() → *PyOpenColorIO.PyOpenColorIO.Config.LookNameIterator*

getLooks() → *PyOpenColorIO.PyOpenColorIO.Config.LookIterator*

getMajorVersion() → int

Get the configuration major version.

getMinorVersion() → int

Get the configuration minor version.

getName() → str

Get/set a name string for the config.

The name string may be used to communicate config update details or similar information to workflows external to OCIO in cases where the config path/filename itself does not provide adequate information.

getNamedTransform(*name: str*) → *PyOpenColorIO.PyOpenColorIO.NamedTransform*

getNamedTransformNames(**args*, ***kwargs*)

Overloaded function.

1. getNamedTransformNames(self: PyOpenColorIO.PyOpenColorIO.Config, visibility: PyOpenColorIO.PyOpenColorIO.NamedTransformVisibility) -> PyOpenColorIO.PyOpenColorIO.Config.NamedTransformNameIterator

2. getNamedTransformNames(self: PyOpenColorIO.PyOpenColorIO.Config) -> PyOpenColorIO.PyOpenColorIO.Config.ActiveNamedTransformNameIterator

getNamedTransforms(**args*, ***kwargs*)

Overloaded function.

1. getNamedTransforms(self: PyOpenColorIO.PyOpenColorIO.Config, visibility: PyOpenColorIO.PyOpenColorIO.NamedTransformVisibility) -> PyOpenColorIO.PyOpenColorIO.Config.NamedTransformIterator

2. getNamedTransforms(self: PyOpenColorIO.PyOpenColorIO.Config) -> PyOpenColorIO.PyOpenColorIO.Config.ActiveNamedTransformIterator

getNumActiveDisplays() → int

Get the number of active displays.

getNumActiveViews() → int

Get the number of active views.

getProcessor(**args*, ***kwargs*)

Overloaded function.

1. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpace: PyOpenColorIO.PyOpenColorIO.ColorSpace, dstColorSpace: PyOpenColorIO.PyOpenColorIO.ColorSpace) -> PyOpenColorIO.PyOpenColorIO.Processor

Get the processor to apply a ColorSpaceTransform from a source to a destination color space.

2. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, srcColorSpace: PyOpenColorIO.PyOpenColorIO.ColorSpace, dstColorSpace: PyOpenColorIO.PyOpenColorIO.ColorSpace) -> PyOpenColorIO.PyOpenColorIO.Processor

3. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, dstColorSpaceName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

---

Note: Names can be colorspace name, role name, or a combination of both.

---

4. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, srcColorSpaceName: str, dstColorSpaceName: str) -> PyOpenColorIO.PyOpenColorIO.Processor

5. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, srcColorSpaceName: str, display: str, view: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

Get the processor to apply a DisplayViewTransform for a display and view. Refer to the Display/View Registration section above for more info on the display and view arguments.

6. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, srcColorSpaceName: str, display: str, view: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

7. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, namedTransform: PyOpenColorIO.PyOpenColorIO.NamedTransform, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

Get the processor to apply a NamedTransform in the specified direction.

8. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, namedTransform: PyOpenColorIO.PyOpenColorIO.NamedTransform, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

9. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, namedTransformName: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

10. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, namedTransformName: str, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

11. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, transform: PyOpenColorIO.PyOpenColorIO.Transform) -> PyOpenColorIO.PyOpenColorIO.Processor

Get the processor for the specified transform.

Not often needed, but will allow for the re-use of atomic OCIO functionality (such as to apply an individual LUT file).

12. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, transform: PyOpenColorIO.PyOpenColorIO.Transform, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

13. getProcessor(self: PyOpenColorIO.PyOpenColorIO.Config, context: PyOpenColorIO.PyOpenColorIO.Context, transform: PyOpenColorIO.PyOpenColorIO.Transform, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection) -> PyOpenColorIO.PyOpenColorIO.Processor

getRoleColorSpace(*roleName: str*) → str

Get the role color space at index.

Return empty string if index is out of range.

getRoleNames() → *PyOpenColorIO.PyOpenColorIO.Config.RoleNameIterator*

getRoles() → *PyOpenColorIO.PyOpenColorIO.Config.RoleColorSpaceIterator*

getSearchPath() → str

getSearchPaths() → *PyOpenColorIO.PyOpenColorIO.Config.SearchPathIterator*

getSharedViews() → *PyOpenColorIO.PyOpenColorIO.Config.SharedViewIterator*

getViewTransform(*name: str*) → *PyOpenColorIO.PyOpenColorIO.ViewTransform*

getViewTransformNames() → *PyOpenColorIO.PyOpenColorIO.Config.ViewTransformNameIterator*

getViewTransforms() → *PyOpenColorIO.PyOpenColorIO.Config.ViewTransformIterator*

getViewingRules() → *PyOpenColorIO.PyOpenColorIO.ViewingRules*

Get read-only version of the viewing rules.

getViews(**args*, ***kwargs*)

Overloaded function.

1. getViews(self: PyOpenColorIO.PyOpenColorIO.Config, display: str) -> PyOpenColorIO.PyOpenColorIO.Config.ViewIterator
2. getViews(self: PyOpenColorIO.PyOpenColorIO.Config, type: PyOpenColorIO.PyOpenColorIO.ViewType, display: str) -> PyOpenColorIO.PyOpenColorIO.Config.ViewForViewTypeIterator
3. getViews(self: PyOpenColorIO.PyOpenColorIO.Config, display: str, colorSpaceName: str) -> PyOpenColorIO.PyOpenColorIO.Config.ViewForColorSpaceIterator

getVirtualDisplayViewColorSpaceName(*view: str*) → str

getVirtualDisplayViewDescription(*view: str*) → str

getVirtualDisplayViewLooks(*view: str*) → str

getVirtualDisplayViewRule(*view: str*) → str

getVirtualDisplayViewTransformName(*view: str*) → str

getVirtualDisplayViews(*display:* PyOpenColorIO.PyOpenColorIO.ViewType) → *PyOpenColorIO.PyOpenColorIO.Config.VirtualViewIterator*

getWorkingDir() → str

hasRole(*role: str*) → bool

Return true if the role has been defined.

hasView(*display: str*, *view: str*) → bool

Determine if a display and view exist.

This returns false if either the display or view doesn't exist. It works regardless of whether the display or view are active, and it works regardless of whether the view is display-defined or if the display has this as a shared view. It will only check config-level shared views if dispName is null. It will not check config level shared views if dispName is not null.

hasVirtualView(*view: str*) → bool

Determine if a virtual view exists.

Methods related to the Virtual Display.

... (See descriptions for the non-virtual methods above.)

The virtual display is the way to incorporate the ICC monitor profile for a user's display into OCIO. The views that are defined for the virtual display are the views that are used to create a new display for an ICC profile. They serve as a kind of template that lets OCIO know how to build the new display.

Typically the views will define a View Transform and set the colorSpaceName to "<USE_DISPLAY_NAME>" so that it will use the display color space with the same name as the

display, in this case corresponding to the ICC profile. This returns false if the virtual view doesn't exist. It works regardless of whether the virtual view is active, and it works regardless of whether the virtual view is display-defined or if the display has this as a shared virtual view.

instantiateDisplayFromICCProfile(*ICCProfileFilepath: str*) → int

Instantiate a new display from a virtual display, using an ICC profile.

On platforms such as Linux, where the SystemMonitors class is not able to obtain a list of ICC profiles from the OS, this method may be used to manually specify a path to an ICC profile.

Will throw if the virtual display definition is missing from the config.

Returns the index of the display.

instantiateDisplayFromMonitorName(*monitorName: str*) → int

Instantiate a new display from a virtual display, using the monitor name.

This method uses the virtual display to create an actual display for the given monitorName. The new display will receive the views from the virtual display.

After the ICC profile is read, a display name will be created by combining the description text from the profile with the monitorName obtained from the OS. Use the SystemMonitors class to obtain the list of monitorName strings for the displays connected to the computer.

A new display color space will also be created using the display name. It will have a from_display_reference transform that is a FileTransform pointing to the ICC profile.

Any instantiated display color spaces for a virtual display are intended to be temporary (i.e. last as long as the current session). By default, they are not saved when writing a config file. If there is a need to make it a permanent color space, it may be desirable to copy the ICC profile somewhere under the config search_path.

Will throw if the config does not have a virtual display or if the monitorName does not exist.

If there is already a display or a display color space with the name monitorName, it will be replaced/updated.

Returns the index of the display.

isArchivable() → bool

Verify if the config is archivable.

A config is not archivable if any of the following are true: – The working directory is not set – It contains FileTransforms with a src outside the working directory – The search path contains paths outside the working directory – The search path contains paths that start with a context variable

Context variables are allowed but the intent is that they may only resolve to paths that are within or below the working directory. This is because the archiving function will only archive files that are within the working directory in order to ensure that if it is later expanded, that it will not create any files outside this directory.

For example, a context variable on the search path intended to contain the name of a sub-directory under the working directory must have the form "./$DIR_NAME" rather than just "$DIR_NAME" to be considered archivable. This is imperfect since there is no way to prevent the context variable from creating a path outside the working dir, but it should at least draw attention to the fact that the archive would fail if used with context vars that try to abuse the intended functionality.

Returns
bool Archivable if true.

isColorSpaceLinear(*colorSpace: str*, *referenceSpaceType:* PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType) → bool

Return true if the specified color space is linear.

The determination of linearity is made with respect to one of the two reference spaces (i.e., either the scene-referred one or the display-referred one). If the reference space type of the color space is the opposite of the requested reference space type, false is returned immediately rather than trying to invoke the default view transform to convert between the reference spaces.

Note: This function relies on heuristics that may sometimes give an incorrect result. For example, if the encoding attribute is not set appropriately or the sampled values fail to detect non-linearity.

The algorithm proceeds as follows: – If the color space isdata attribute is true, return false. – If the reference space type of the color space differs from the requested reference space type, return false. – If the color space's encoding attribute is present, return true if it matches the expected reference space type (i.e., "scene-linear" for REFERENCE_SPACE_SCENE or "display-linear" for REFERENCE_SPACE_DISPLAY) and false otherwise. – If the color space has no to_reference or from_reference transform, return true. – Evaluate several points through the color space's transform and check if the output only differs by a scale factor (which may be different per channel, e.g. allowing an arbitrary matrix transform, with no offset).

Note that the encoding test happens before the sampled value test to give config authors ultimate control over the linearity determination. For example, they could set the encoding attribute to indicate linearity if they want to ignore some areas of non-linearity (e.g., at extreme values). Or they could set it to indicate that a color space should not be considered linear, even if it is, in a mathematical sense.

Parameters

- **colorSpace** – Color space to evaluate.
- **referenceSpaceType** – Evaluate linearity with respect to the specified reference space (either scene-referred or display-referred).

isColorSpaceUsed(*name: str*) → bool

Return true if the color space is used by a transform, a role, or a look.

Note: Name must be the canonical name.

---

isDisplayTemporary(*display: str*) → bool

isInactiveColorSpace(*colorspace: str*) → bool

Return true if the color space name is present in the inactive_colorspaces list.

isStrictParsingEnabled() → bool

isViewShared(*display: str*, *view: str*) → bool

Check if a view within a given display is referencing one of the config's shared views.

Methods related to displays and views.

The following methods only manipulate active displays and views. Active displays and views are defined from an env. variable or from the config file.

Looks is a potentially comma (or colon) delimited list of lookNames, Where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either)

Add shared view (or replace existing one with same name). Shared views are defined at config level and can be referenced by several displays. Either provide a view transform and a display color space or just a color space (and a null view transform). Looks, rule and description are optional, they can be null or empty.

Shared views using a view transform may use the token <USE_DISPLAY_NAME> for the color space (see `OCIO_VIEW_USE_DISPLAY_NAME`). In that case, when the view is referenced in a display, the display color

space that is used will be the one matching the display name. In other words, the view will be customized based on the display it is used in. Config::validate will throw if the config does not contain the matching display color space.

isVirtualViewShared(*view: str*) → bool

Check if a given virtual view is referencing one of the config's shared views.

loadEnvironment() → None

Initialize the environment/context variables in the Config's Context.

parseColorSpaceFromString(*str: str*) → str

Given the specified string, get the longest, right-most, colorspace substring that appears.

- If strict parsing is enabled, and no color space is found, return an empty string.
- If strict parsing is disabled, return ROLE_DEFAULT (if defined).
- If the default role is not defined, return an empty string.

removeActiveDisplay(*display: str*) → None

Remove a single display. Will throw if the display is not present.

removeActiveView(*view: str*) → None

Remove a single view. Will throw if the view is not present.

removeColorSpace(*name: str*) → None

Remove a color space from the configuration.

Note: It does not throw an exception. Name must be the canonical name. If a role name or alias is provided or if the name is not in the config, nothing is done.

Note: Removing a color space from a Config does not affect any ColorSpaceSet sets that have already been created.

---

removeDisplayView(*display: str*, *view: str*) → None

Remove the view and the display if no more views.

It does not remove the associated color space. If the view name is a shared view, it only removes the reference to the view from the display but the shared view, remains in the config.

Will throw if the view does not exist.

removeSharedView(*view: str*) → None

Remove a shared view. Will throw if the view does not exist.

removeVirtualDisplayView(*view: str*) → None

Remove the view from the virtual display.

serialize(**args*, ***kwargs*)

Overloaded function.

1. serialize(self: PyOpenColorIO.PyOpenColorIO.Config, fileName: str) -> None
2. serialize(self: PyOpenColorIO.PyOpenColorIO.Config) -> str

Returns the string representation of the Config in YAML text form.

This is typically stored on disk in a file with the extension .ocio. NB: This does not validate the config. Applications should validate before serializing.

setActiveDisplays(*displays: str*) → None

The Active Displays list allows end users, config authors, and client apps to filter and reorder of the list of displays available in a user-interface. The list may be left empty to indicate all displays are active.

The first active display is the config's Default Display.

If the active list would remove all displays from a config, it is ignored (though the config won't validate).

When serialized in the config, commas are used as separators. However, if a display name contains a comma, the name will be enclosed in quotes so its comma is not a separator.

The OCIO_ACTIVE_DISPLAYS environment variable will override the active list specified in the config file as well as any modifications made by the client app. These functions only get and set what is in the config object and do not take into account the override and thus may not represent the actual user experience.Set all active displays at once as a comma or colon delimited string. This replaces any previous contents of the list.

setActiveViews(*views: str*) → None

The Active Views list allows end users, config authors, and client apps to filter and reorder of the list of views available in a user-interface. The list may be left empty to indicate all views are active.

The first active view for a display is its Default View.

If the active list would remove all views from a display, the list is ignored for that display and all views are shown for it.

When serialized in the config, commas are used as separators. However, if a view name contains a comma, the name will be enclosed in quotes so its comma is not a separator.

The OCIO_ACTIVE_VIEWS environment variable will override the active list specified in the config file as well as any modifications made by the client app. These functions only get and set what is in the config object and do not take into account the override and thus may not represent the actual user experience.Set all active views at once as a comma or colon delimited string. This replaces any previous contents of the list.

setConfigIOProxy(*ciop: OpenColorIO_v2_5::ConfigIOProxy*) → None

Set the ConfigIOProxy object used to provision the config and LUTs from somewhere other than the file system. (This is set on the config's embedded Context object.)

setDefaultLumaCoefs(*rgb: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

These should be normalized (sum to 1.0 exactly).

setDefaultViewTransformName(*name: str*) → None

setDescription(*description: str*) → None

setDisplayTemporary(*display: str*, *isTemporary: bool*) → None

setEnvironmentMode(*mode:* PyOpenColorIO.PyOpenColorIO.EnvironmentMode) → None

The EnvironmentMode controls the behavior of loadEnvironment.

- ENV_ENVIRONMENT_LOAD_PREDEFINED - Only update vars already added to the Context.
- ENV_ENVIRONMENT_LOAD_ALL - Load all env. vars into the Context.

Note: Loading ALL the env. vars may reduce performance and reduce cache efficiency.

Client programs generally will not use these methods because the EnvironmentMode is set automatically when a:ref:*Config* is loaded. If the Config has an "environment" section, the mode is set to LOAD_PREDEFINED, and otherwise set to LOAD_ALL.

setFamilySeparator(*separator: str*) → None

Set the family separator.

Succeeds if the characters is null or a valid character from the ASCII table i.e. from value 32 (i.e. space) to 126 (i.e. '~'); otherwise, it throws an exception.

setFileRules(*fileRules:* PyOpenColorIO.PyOpenColorIO.FileRules) → None

Set file rules.

Note: The argument is cloned.

---

setInactiveColorSpaces(*inactiveColorSpaces: str*) → None

Set/get a list of inactive color space or named transform names.

Notes: - List can contain color space and/or named transform names. - The inactive spaces are color spaces that should not appear in application menus. - These color spaces will still work in Config::getProcessor calls. - The argument is a comma-delimited string. A null or empty string empties the list. - The environment variable OCIO_INACTIVE_COLORSPACES may also be used to set the inactive color space list. - The env. var. takes precedence over the inactive_colorspaces list in the config file. - Setting the list via the API takes precedence over either the env. var. or the config file list.

setMajorVersion(*major: SupportsInt*) → None

Set the configuration major version.

Throws if it is not supported. Resets minor to the most recent minor for the given major.

setMinorVersion(*minor: SupportsInt*) → None

Set the configuration minor version. Throws if it is not supported for the current major.

setName(*name: str*) → None

setProcessorCacheFlags(*flags:* PyOpenColorIO.PyOpenColorIO.ProcessorCacheFlags) → None

Control the caching of processors in the config instance. By default, caching is on. The flags allow turning caching off entirely or only turning it off if dynamic properties are being used by the processor.

setRole(*role: str*, *colorSpaceName: str*) → None

Methods related to Roles.

A role allows a config author to indicate that a given color space should be used for a particular purpose.

Role names may be passed to most functions that accept color space names, such as getColorSpace. So for example, you may find the name of the color space assigned to the scene_linear role by getting the color space object for "scene_linear" and then calling getName on the color space object... note::
Setting the *colorSpaceName* name to a null string unsets it.

setSearchPath(*path: str*) → None

Set all search paths as a concatenated string, use ':' to separate the paths.

See addSearchPath for a more robust and platform-agnostic method of setting the search paths.

setStrictParsingEnabled(*enabled: bool*) → None

setVersion(*major: SupportsInt*, *minor: SupportsInt*) → None

Set the configuration major and minor versions. Throws if version is not supported.

setViewingRules(*ViewingRules:* PyOpenColorIO.PyOpenColorIO.ViewingRules) → None

Set viewing rules.

Note: The argument is cloned.

---

setWorkingDir(*dirName: str*) → None

The working directory defaults to the location of the config file. It is used to convert any relative paths to absolute. If no search paths have been set, the working directory will be used as the fallback search path. No environment (context) variables may be used in the working directory.

upgradeToLatestVersion() → None

Allows an older config to be serialized as the current version.

validate() → None

Performs a thorough validation for the most common user errors.

This will throw an exception if the config is malformed. The most common error occurs when references are made to colorspaces that do not exist.

class PyOpenColorIO.Config.**EnvironmentVarNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.EnvironmentVarNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**SearchPathIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.SearchPathIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ColorSpaceNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ColorSpaceNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ColorSpaceIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ColorSpaceIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

class PyOpenColorIO.Config.**ActiveColorSpaceNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ActiveColorSpaceNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ActiveColorSpaceIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ActiveColorSpaceIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.ColorSpace*

class PyOpenColorIO.Config.**RoleNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.RoleNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**RoleColorSpaceIterator**

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.RoleColorSpaceIterator*

len(*self*) → int

next(*self*) → tuple

class PyOpenColorIO.Config.**DisplayIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.DisplayIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**SharedViewIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.SharedViewIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ViewIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ViewIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ViewForColorSpaceIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ViewForColorSpaceIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**LookNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.LookNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**LookIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.Look*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.LookIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.Look*

class PyOpenColorIO.Config.**ViewTransformNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ViewTransformNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ViewTransformIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.ViewTransform*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ViewTransformIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.ViewTransform*

class PyOpenColorIO.Config.**NamedTransformNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.NamedTransformNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**NamedTransformIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.NamedTransform*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.NamedTransformIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.NamedTransform*

class PyOpenColorIO.Config.**ActiveNamedTransformNameIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ActiveNamedTransformNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Config.**ActiveNamedTransformIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.NamedTransform*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Config.ActiveNamedTransformIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.NamedTransform*

C++

ConstConfigRcPtr OpenColorIO_v2_5::**GetCurrentConfig**()

Get the current configuration.

void OpenColorIO_v2_5::**SetCurrentConfig**(const *ConstConfigRcPtr* &config)

Set the current configuration. This will then store a copy of the specified config.

class **Config**

A config defines all the color spaces to be available at runtime.

The color configuration (Config) is the main object for interacting with this library. It encapsulates all of the information necessary to use customized ColorSpaceTransform and DisplayViewTransform operations.

See the user-guide for more information on selecting, creating, and working with custom color configurations.

For applications interested in using only one color config at a time (this is the vast majority of apps), their API would traditionally get the global configuration and use that, as opposed to creating a new one. This simplifies the use case for plugins and bindings, as it alleviates the need to pass around configuration handles.

An example of an application where this would not be sufficient would be a multi-threaded image proxy server (daemon), which wished to handle multiple show configurations in a single process concurrently. This app would need to keep multiple configurations alive, and to manage them appropriately.

Roughly speaking, a novice user should select a default configuration that most closely approximates the use case (animation, visual effects, etc.), and set the :envvar:`OCIO` environment variable to point at the root of that configuration.

See developers-usageexamples

Note: Initialization using environment variables is typically preferable in a multi-app ecosystem, as it allows all applications to be consistently configured.

Public Functions

ConfigRcPtr **createEditableCopy**() const

unsigned int **getMajorVersion**() const

Get the configuration major version.

void **setMajorVersion**(unsigned int major)

Set the configuration major version.

Throws if it is not supported. Resets minor to the most recent minor for the given major.

unsigned int **getMinorVersion**() const

Get the configuration minor version.

void **setMinorVersion**(unsigned int minor)

Set the configuration minor version. Throws if it is not supported for the current major.

void **setVersion**(unsigned int major, unsigned int minor)

Set the configuration major and minor versions. Throws if version is not supported.

void **upgradeToLatestVersion**() noexcept

Allows an older config to be serialized as the current version.

void **validate**() const

Performs a thorough validation for the most common user errors.

This will throw an exception if the config is malformed. The most common error occurs when references are made to colorspaces that do not exist.

const char ***getName**() const noexcept

Get/set a name string for the config.

The name string may be used to communicate config update details or similar information to workflows external to OCIO in cases where the config path/filename itself does not provide adequate information.

void **setName**(const char *name) noexcept

char **getFamilySeparator**() const

Get the family separator.

A single character used to separate the family string into tokens for use in hierarchical menus. Defaults to '/'.

void **setFamilySeparator**(char separator)

Set the family separator.

Succeeds if the characters is null or a valid character from the ASCII table i.e. from value 32 (i.e. space) to 126 (i.e. '~'); otherwise, it throws an exception.

const char ***getDescription**() const

void **setDescription**(const char *description)

void **serialize**(std::ostream &os) const

Returns the string representation of the Config in YAML text form.

This is typically stored on disk in a file with the extension .ocio. NB: This does not validate the config. Applications should validate before serializing.

const char ***getCacheID**() const

This will produce a hash of the all colorspace definitions, etc. All external references, such as files used in FileTransforms, etc., will be incorporated into the cacheID. While the contents of the files are not read, the file system is queried for relevant information (mtime, inode) so that the config's cacheID will change when the underlying luts are updated.

If a context is not provided, the current Context will be used.

If a null context is provided, file references will not be taken into account (this is essentially a hash of Config::serialize).

const char ***getCacheID**(const *ConstContextRcPtr* &context) const

ConstContextRcPtr **getCurrentContext**() const

void **addEnvironmentVar**(const char *name, const char *defaultValue)

Add (or update) an environment variable with a default value. But it removes it if the default value is null.

int **getNumEnvironmentVars**() const

const char ***getEnvironmentVarNameByIndex**(int index) const

const char ***getEnvironmentVarDefault**(const char *name) const

void **clearEnvironmentVars**()

void **setEnvironmentMode**(*EnvironmentMode* mode) noexcept

The EnvironmentMode controls the behavior of loadEnvironment.

- ENV_ENVIRONMENT_LOAD_PREDEFINED - Only update vars already added to the Context.
- ENV_ENVIRONMENT_LOAD_ALL - Load all env. vars into the Context.

Client programs generally will not use these methods because the EnvironmentMode is set automatically when a Config is loaded. If the Config has an "environment" section, the mode is set to LOAD_PREDEFINED, and otherwise set to LOAD_ALL.

Note: Loading ALL the env. vars may reduce performance and reduce cache efficiency.

EnvironmentMode **getEnvironmentMode**() const noexcept

void **loadEnvironment**() noexcept

Initialize the environment/context variables in the Config's Context.

const char ***getSearchPath**() const

void **setSearchPath**(const char *path)

Set all search paths as a concatenated string, use ':' to separate the paths.

See addSearchPath for a more robust and platform-agnostic method of setting the search paths.

int **getNumSearchPaths**() const

const char ***getSearchPath**(int index) const

Get a search path from the list.

The paths are in the order they will be searched (that is, highest to lowest priority).

void **clearSearchPaths**()

void **addSearchPath**(const char *path)

Add a single search path to the end of the list.

Paths may be either absolute or relative. Relative paths are relative to the working directory. Forward slashes will be normalized to reverse for Windows. Environment (context) variables may be used in paths.

const char ***getWorkingDir**() const

void **setWorkingDir**(const char *dirname)

The working directory defaults to the location of the config file. It is used to convert any relative paths to absolute. If no search paths have been set, the working directory will be used as the fallback search path. No environment (context) variables may be used in the working directory.

ColorSpaceSetRcPtr **getColorSpaces**(const char *category) const

Get all active color spaces having a specific category in the order they appear in the config file.

Note: If the category is null or empty, the method returns all the active color spaces like Config::getNumColorSpaces and Config::getColorSpaceNameByIndex do.

Note: It's worth noticing that the method returns a copy of the selected color spaces decoupling the result from the config. Hence, any changes on the config do not affect the existing color space sets, and vice-versa.

---

int **getNumColorSpaces**(*SearchReferenceSpaceType* searchReferenceType, *ColorSpaceVisibility* visibility) const

Work on the color spaces selected by the reference color space type and visibility.

const char ***getColorSpaceNameByIndex**(*SearchReferenceSpaceType* searchReferenceType, *ColorSpaceVisibility* visibility, int index) const

Work on the color spaces selected by the reference color space type and visibility (active or inactive).

Return empty for invalid index.

int **getNumColorSpaces**() const

Work on the active color spaces only.

Note: Only works from the list of active color spaces.

---

const char ***getColorSpaceNameByIndex**(int index) const

Work on the active color spaces only and return null for invalid index.

Note: Only works from the list of active color spaces.

---

int **getIndexForColorSpace**(const char *name) const

Get an index from the active color spaces only and return -1 if the name is not found.

Note: The fcn accepts either a color space name, role name, or alias. (Color space names take precedence over roles.)

ConstColorSpaceRcPtr **getColorSpace**(const char *name) const

Get the color space from all the color spaces (i.e. active and inactive) and return null if the name is not found.

Note: The fcn accepts either a color space name, role name, or alias. (Color space names take precedence over roles.)

---

const char ***getCanonicalName**(const char *name) const

Accepts an alias, role name, named transform name, or color space name and returns the color space name or the named transform name.

void **addColorSpace**(const *ConstColorSpaceRcPtr* &cs)

Add a color space to the configuration.

Note: If another color space is already present with the same name, this will overwrite it. This stores a copy of the specified color space.

Note: Adding a color space to a Config does not affect any ColorSpaceSet sets that have already been created.

---

void **removeColorSpace**(const char *name)

Remove a color space from the configuration.

Note: It does not throw an exception. Name must be the canonical name. If a role name or alias is provided or if the name is not in the config, nothing is done.

Note: Removing a color space from a Config does not affect any ColorSpaceSet sets that have already been created.

---

bool **isColorSpaceUsed**(const char *name) const noexcept

Return true if the color space is used by a transform, a role, or a look.

Note: Name must be the canonical name.

---

void **clearColorSpaces**()

Remove all the color spaces from the configuration.

Note: Removing color spaces from a Config does not affect any ColorSpaceSet sets that have already been created.

---

void **setInactiveColorSpaces**(const char *inactiveColorSpaces)

Set/get a list of inactive color space or named transform names.

Notes:

- List can contain color space and/or named transform names.
- The inactive spaces are color spaces that should not appear in application menus.
- These color spaces will still work in Config::getProcessor calls.
- The argument is a comma-delimited string. A null or empty string empties the list.
- The environment variable OCIO_INACTIVE_COLORSPACES may also be used to set the inactive color space list.
- The env. var. takes precedence over the inactive_colorspaces list in the config file.
- Setting the list via the API takes precedence over either the env. var. or the config file list.

const char ***getInactiveColorSpaces**() const

bool **isInactiveColorSpace**(const char *colorspace) const noexcept

Return true if the color space name is present in the inactive_colorspaces list.

bool **isColorSpaceLinear**(const char *colorSpace, *ReferenceSpaceType* referenceSpaceType) const

Return true if the specified color space is linear.

The determination of linearity is made with respect to one of the two reference spaces (i.e., either the scene-referred one or the display-referred one). If the reference space type of the color space is the opposite of the requested reference space type, false is returned immediately rather than trying to invoke the default view transform to convert between the reference spaces.

Note: This function relies on heuristics that may sometimes give an incorrect result. For example, if the encoding attribute is not set appropriately or the sampled values fail to detect non-linearity.

The algorithm proceeds as follows: — If the color space isdata attribute is true, return false. — If the reference space type of the color space differs from the requested reference space type, return false. — If the color space's encoding attribute is present, return true if it matches the expected reference space type (i.e., "scene-linear" for REFERENCE_SPACE_SCENE or "display-linear" for REFERENCE_SPACE_DISPLAY) and false otherwise. — If the color space has no to_reference or from_reference transform, return true. — Evaluate several points through the color space's transform and check if the output only differs by a scale factor (which may be different per channel, e.g. allowing an arbitrary matrix transform, with no offset).

Note that the encoding test happens before the sampled value test to give config authors ultimate control over the linearity determination. For example, they could set the encoding attribute to indicate linearity if

they want to ignore some areas of non-linearity (e.g., at extreme values). Or they could set it to indicate that a color space should not be considered linear, even if it is, in a mathematical sense.

Parameters

- **colorSpace** – Color space to evaluate.
- **referenceSpaceType** – Evaluate linearity with respect to the specified reference space (either scene-referred or display-referred).

void **setRole**(const char *role, const char *colorSpaceName)

Methods related to Roles.

A role allows a config author to indicate that a given color space should be used for a particular purpose.

Role names may be passed to most functions that accept color space names, such as getColorSpace. So for example, you may find the name of the color space assigned to the scene_linear role by getting the color space object for "scene_linear" and then calling getName on the color space object.

Note: Setting the `colorSpaceName` name to a null string unsets it.

---

int **getNumRoles**() const

bool **hasRole**(const char *role) const

Return true if the role has been defined.

const char ***getRoleName**(int index) const

Get the role name at index, this will return values like 'scene_linear', 'compositing_log'.

Return empty string if index is out of range.

const char ***getRoleColorSpace**(int index) const

Get the role color space at index.

Return empty string if index is out of range.

const char ***getRoleColorSpace**(const char *roleName) const noexcept

Get the color space name used for the specified role.

Return an empty string if the role is not present

bool **isViewShared**(const char *dispName, const char *viewName) const

Check if a view within a given display is referencing one of the config's shared views.

Methods related to displays and views.

The following methods only manipulate active displays and views. Active displays and views are defined from an env. variable or from the config file.

Looks is a potentially comma (or colon) delimited list of lookNames, Where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either)

Add shared view (or replace existing one with same name). Shared views are defined at config level and can be referenced by several displays. Either provide a view transform and a display color space or just a color space (and a null view transform). Looks, rule and description are optional, they can be null or empty.

Shared views using a view transform may use the token <USE_DISPLAY_NAME> for the color space (see :c:var:`OCIO_VIEW_USE_DISPLAY_NAME`). In that case, when the view is referenced in a display, the display color space that is used will be the one matching the display name. In other words, the view will be customized based on the display it is used in. Config::validate will throw if the config does not contain the matching display color space.

void **addSharedView**(const char *view, const char *viewTransformName, const char *colorSpaceName, const char *looks, const char *ruleName, const char *description)

Will throw if view or colorSpaceName are null or empty.

void **removeSharedView**(const char *view)

Remove a shared view. Will throw if the view does not exist.

void **clearSharedViews**()

Clear all shared views. This will throw if any displays are still using the shared views.

const char ***getDefaultDisplay**() const

int **getNumDisplays**() const

const char ***getDisplay**(int index) const

Will return "" if the index is invalid.

const char ***getDefaultView**(const char *display) const

const char ***getDefaultView**(const char *display, const char *colorspaceName) const

int **getNumViews**(const char *display) const

Return the number of active views attached to the display including the number of shared views if any. Return 0 if display does not exist.

const char ***getView**(const char *display, int index) const

int **getNumViews**(const char *display, const char *colorspaceName) const

If the config has ViewingRules, get the number of active Views for this colorspace. (If there are no rules, it returns all of them.)

const char ***getView**(const char *display, const char *colorspaceName, int index) const

const char ***getDisplayViewTransformName**(const char *display, const char *view) const

Returns the view_transform attribute of the (display, view) pair. View can be a shared view of the display. If display is null or empty, config shared views are used.

const char ***getDisplayViewColorSpaceName**(const char *display, const char *view) const

Returns the colorspace attribute of the (display, view) pair. (Note that this may be either a color space or a display color space.)

const char ***getDisplayViewLooks**(const char *display, const char *view) const

Returns the looks attribute of a (display, view) pair.

const char ***getDisplayViewRule**(const char *display, const char *view) const noexcept

Returns the rule attribute of a (display, view) pair.

const char ***getDisplayViewDescription**(const char *display, const char *view) const noexcept

Returns the description attribute of a (display, view) pair.

bool **hasView**(const char *dispName, const char *viewName) const

Determine if a display and view exist.

This returns false if either the display or view doesn't exist. It works regardless of whether the display or view are active, and it works regardless of whether the view is display-defined or if the display has this as a shared view. It will only check config-level shared views if dispName is null. It will not check config level shared views if dispName is not null.

void **addDisplayView**(const char *display, const char *view, const char *colorSpaceName, const char *looks)

For the (display, view) pair, specify which color space and look to use. If a look is not desired, then just pass a null or empty string.

void **addDisplayView**(const char *display, const char *view, const char *viewTransformName, const char *colorSpaceName, const char *looks, const char *ruleName, const char *description)

For the (display, view) pair, specify the color space or alternatively specify the view transform and display color space. The looks, viewing rule, and description are optional. Pass a null or empty string for any optional arguments. If the view already exists, it is replaced.

Will throw if:

- Display, view or colorSpace are null or empty.
- Display already has a shared view with the same name.

void **addDisplaySharedView**(const char *display, const char *sharedView)

Add a (reference to a) shared view to a display.

The shared view must be part of the config. See Config::addSharedView

This will throw if:

- Display or view are null or empty.
- Display already has a view (shared or not) with the same name.

void **removeDisplayView**(const char *display, const char *view)

Remove the view and the display if no more views.

It does not remove the associated color space. If the view name is a shared view, it only removes the reference to the view from the display but the shared view, remains in the config.

Will throw if the view does not exist.

void **clearDisplays**()

Clear all the displays.

bool **hasVirtualView**(const char *viewName) const

Determine if a virtual view exists.

Methods related to the Virtual Display.

... (See descriptions for the non-virtual methods above.)

The virtual display is the way to incorporate the ICC monitor profile for a user's display into OCIO. The views that are defined for the virtual display are the views that are used to create a new display for an ICC profile. They serve as a kind of template that lets OCIO know how to build the new display.

Typically the views will define a View Transform and set the colorSpaceName to "<USE_DISPLAY_NAME>" so that it will use the display color space with the same name as the display, in this case corresponding to the ICC profile. This returns false if the virtual view doesn't exist. It works regardless of whether the virtual view is active, and it works regardless of whether the virtual view is display-defined or if the display has this as a shared virtual view.

bool **isVirtualViewShared**(const char *viewName) const

Check if a given virtual view is referencing one of the config's shared views.

void **addVirtualDisplayView**(const char *view, const char *viewTransformName, const char *colorSpaceName, const char *looks, const char *ruleName, const char *description)

void **addVirtualDisplaySharedView**(const char *sharedView)

int **getVirtualDisplayNumViews**(*ViewType* type) const noexcept

Get the number of views associated to the virtual display.

const char ***getVirtualDisplayView**(*ViewType* type, int index) const noexcept

Get the view name at a specific index.

const char ***getVirtualDisplayViewTransformName**(const char *view) const noexcept

const char ***getVirtualDisplayViewColorSpaceName**(const char *view) const noexcept

const char ***getVirtualDisplayViewLooks**(const char *view) const noexcept

const char ***getVirtualDisplayViewRule**(const char *view) const noexcept

const char ***getVirtualDisplayViewDescription**(const char *view) const noexcept

void **removeVirtualDisplayView**(const char *view) noexcept

Remove the view from the virtual display.

void **clearVirtualDisplay**() noexcept

Clear the virtual display.

int **instantiateDisplayFromMonitorName**(const char *monitorName)

Instantiate a new display from a virtual display, using the monitor name.

This method uses the virtual display to create an actual display for the given monitorName. The new display will receive the views from the virtual display.

After the ICC profile is read, a display name will be created by combining the description text from the profile with the monitorName obtained from the OS. Use the SystemMonitors class to obtain the list of monitorName strings for the displays connected to the computer.

A new display color space will also be created using the display name. It will have a from_display_reference transform that is a FileTransform pointing to the ICC profile.

Any instantiated display color spaces for a virtual display are intended to be temporary (i.e. last as long as the current session). By default, they are not saved when writing a config file. If there is a need to make it a permanent color space, it may be desirable to copy the ICC profile somewhere under the config search_path.

Will throw if the config does not have a virtual display or if the monitorName does not exist.

If there is already a display or a display color space with the name monitorName, it will be replaced/updated.

Returns the index of the display.

int **instantiateDisplayFromICCProfile**(const char *ICCProfileFilepath)

Instantiate a new display from a virtual display, using an ICC profile.

On platforms such as Linux, where the SystemMonitors class is not able to obtain a list of ICC profiles from the OS, this method may be used to manually specify a path to an ICC profile.

Will throw if the virtual display definition is missing from the config.

Returns the index of the display.

void **setActiveDisplays**(const char *displays)

The Active Displays list allows end users, config authors, and client apps to filter and reorder of the list of displays available in a user-interface. The list may be left empty to indicate all displays are active.

The first active display is the config's Default Display.

If the active list would remove all displays from a config, it is ignored (though the config won't validate).

When serialized in the config, commas are used as separators. However, if a display name contains a comma, the name will be enclosed in quotes so its comma is not a separator.

The OCIO_ACTIVE_DISPLAYS environment variable will override the active list specified in the config file as well as any modifications made by the client app. These functions only get and set what is in the config object and do not take into account the override and thus may not represent the actual user experience. Set all active displays at once as a comma or colon delimited string. This replaces any previous contents of the list.

const char ***getActiveDisplays**() const

Get a string with all active displays (as it would appear in a config file). Commas are always used as the separator.

int **getNumActiveDisplays**() const

Get the number of active displays.

const char ***getActiveDisplay**(int index) const

Get a single active display, by index. Returns nullptr if the index is out of range.

void **addActiveDisplay**(const char *display)

Add a single active display to the end of the list. If the display is already present, no action is taken.

void **removeActiveDisplay**(const char *display)

Remove a single display. Will throw if the display is not present.

void **clearActiveDisplays**()

Clear the active displays list.

void **setActiveViews**(const char *views)

The Active Views list allows end users, config authors, and client apps to filter and reorder of the list of views available in a user-interface. The list may be left empty to indicate all views are active.

The first active view for a display is its Default View.

If the active list would remove all views from a display, the list is ignored for that display and all views are shown for it.

When serialized in the config, commas are used as separators. However, if a view name contains a comma, the name will be enclosed in quotes so its comma is not a separator.

The OCIO_ACTIVE_VIEWS environment variable will override the active list specified in the config file as well as any modifications made by the client app. These functions only get and set what is in the config object and do not take into account the override and thus may not represent the actual user experience. Set all active views at once as a comma or colon delimited string. This replaces any previous contents of the list.

const char ***getActiveViews**() const

Get a string with all active views (as it would appear in a config file). Commas are always used as the separator.

int **getNumActiveViews**() const

Get the number of active views.

const char ***getActiveView**(int index) const

Get a single active view, by index. Returns nullptr if the index is out of range.

void **addActiveView**(const char *view)

Add a single active view to the end of the list. If the view is already present, no action is taken.

void **removeActiveView**(const char *view)

Remove a single view. Will throw if the view is not present.

void **clearActiveViews**()

Clear the active views list.

int **getNumDisplaysAll**() const noexcept

Get all displays in the config, ignoring the active_displays list.

const char ***getDisplayAll**(int index) const noexcept

int **getDisplayAllByName**(const char*) const noexcept

bool **isDisplayTemporary**(int index) const noexcept

Will be true for a display that was instantiated from a virtual display. These displays are intended to be temporary (i.e. for the current session) and are not saved to a config file.

void **setDisplayTemporary**(int index, bool isTemporary) noexcept

Allows setting the flag that controls whether a display is temporary. This may be helpful, for example, to share a config with a temporary instantiated display with an OFX plug-in.

int **getNumViews**(*ViewType* type, const char *display) const

Get either the shared or display-defined views for a display. The active_views list is ignored. Passing a null or empty display (with type=VIEW_SHARED) returns the contents of the shared_views section of the config. Return 0 if display does not exist.

const char ***getView**(*ViewType* type, const char *display, int index) const

ConstViewingRulesRcPtr **getViewingRules**() const noexcept

Get read-only version of the viewing rules.

void **setViewingRules**(*ConstViewingRulesRcPtr* viewingRules)

Set viewing rules.

Note: The argument is cloned.

---

void **getDefaultLumaCoefs**(double *rgb) const

Get the default coefficients for computing luma.

Note: There is no "1 size fits all" set of luma coefficients. (The values are typically different for each colorspace, and the application of them may be nonsensical depending on the intensity coding anyways). Thus, the 'right' answer is to make these functions on the ColorSpace class. However, it's often useful to have a config-wide default so here it is. We will add the colorspace specific luma call if/when another client is interesting in using it.

---

void **setDefaultLumaCoefs**(const double *rgb)

These should be normalized (sum to 1.0 exactly).

ConstLookRcPtr **getLook**(const char *name) const

int **getNumLooks**() const

const char ***getLookNameByIndex**(int index) const

void **addLook**(const *ConstLookRcPtr* &look)

void **clearLooks**()

int **getNumViewTransforms**() const noexcept

ConstViewTransformRcPtr **getViewTransform**(const char *name) const noexcept

const char ***getViewTransformNameByIndex**(int i) const noexcept

void **addViewTransform**(const *ConstViewTransformRcPtr* &viewTransform)

ConstViewTransformRcPtr **getDefaultSceneToDisplayViewTransform**() const

This view transform is the one that will be used by default if a ColorSpaceTransform is needed between a scene-referred and display-referred color space. The config author may specify a transform to use via the default_view_transform entry in the config. If that is not present, or does not return a valid view transform from the scene-referred connection space, the fall-back is to use the first valid view transform in the config. Returns a null ConstTransformRcPtr if there isn't one.

const char ***getDefaultViewTransformName**() const noexcept

Get or set the default_view_transform string from the config.

Note that if this is not the name of a valid view transform from the scene-referred connection space, it will be ignored.

void **setDefaultViewTransformName**(const char *defaultName) noexcept

void **clearViewTransforms**()

int **getNumNamedTransforms**(*NamedTransformVisibility* visibility) const noexcept

Work on the named transforms selected by visibility.

Methods related to named transforms.

const char ***getNamedTransformNameByIndex**(*NamedTransformVisibility* visibility, int index) const noexcept

Work on the named transforms selected by visibility (active or inactive).

Return an empty string for invalid index.

int **getNumNamedTransforms**() const noexcept

Work on the active named transforms only.

const char ***getNamedTransformNameByIndex**(int index) const noexcept

Work on the active named transforms only and return an empty string for invalid index.

int **getIndexForNamedTransform**(const char *name) const noexcept

Get an index from the active named transforms only and return -1 if the name is not found.

ConstNamedTransformRcPtr **getNamedTransform**(const char *name) const noexcept

Get the named transform from all the named transforms (i.e. active and inactive) and return null if the name is not found.

void **addNamedTransform**(const *ConstNamedTransformRcPtr* &namedTransform)

Add or replace named transform.

Note: Throws if namedTransform is null, name is missing, or no transform is set. Also throws if the name or the aliases conflict with names or aliases already in the config.

---

void **removeNamedTransform**(const char *name)

Remove a named transform. (Does nothing if name is not found.)

void **clearNamedTransforms**()

Clear all named transforms.

ConstFileRulesRcPtr **getFileRules**() const noexcept

Get read-only version of the file rules.

void **setFileRules**(*ConstFileRulesRcPtr* fileRules)

Set file rules.

Note: The argument is cloned.

---

const char ***getColorSpaceFromFilepath**(const char *filePath) const

Get the color space of the first rule that matched filePath. (For v1 configs, this is equivalent to calling parseColorSpaceFromString with strictparsing set to false.)

const char ***getColorSpaceFromFilepath**(const char *filePath, size_t &ruleIndex) const

Most applications will use the preceding method, but this method may be used for applications that want to know which was the highest priority rule to match filePath. FileRules::getNumCustomKeys and custom keys methods may then be used to get additional information about the matching rule.

bool **filepathOnlyMatchesDefaultRule**(const char *filePath) const

Returns true if the only rule matched by filePath is the default rule. This is a convenience method for applications that want to require the user to manually choose a color space when strictParsing is true and no other rules match.

const char ***parseColorSpaceFromString**(const char *str) const

Given the specified string, get the longest, right-most, colorspace substring that appears.

- If strict parsing is enabled, and no color space is found, return an empty string.
- If strict parsing is disabled, return ROLE_DEFAULT (if defined).
- If the default role is not defined, return an empty string.

bool **isStrictParsingEnabled**() const

void **setStrictParsingEnabled**(bool enabled)

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const *ConstColorSpaceRcPtr* &srcColorSpace, const *ConstColorSpaceRcPtr* &dstColorSpace) const

Get the processor to apply a ColorSpaceTransform from a source to a destination color space.

ConstProcessorRcPtr **getProcessor**(const *ConstColorSpaceRcPtr* &srcColorSpace, const *ConstColorSpaceRcPtr* &dstColorSpace) const

ConstProcessorRcPtr **getProcessor**(const char *srcColorSpaceName, const char *dstColorSpaceName) const

Note: Names can be colorspace name, role name, or a combination of both.

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const char *srcColorSpaceName, const char *dstColorSpaceName) const

ConstProcessorRcPtr **getProcessor**(const char *srcColorSpaceName, const char *display, const char *view, *TransformDirection* direction) const

Get the processor to apply a DisplayViewTransform for a display and view. Refer to the Display/View Registration section above for more info on the display and view arguments.

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const char *srcColorSpaceName, const char *display, const char *view, *TransformDirection* direction) const

ConstProcessorRcPtr **getProcessor**(const *ConstNamedTransformRcPtr* &namedTransform, *TransformDirection* direction) const

Get the processor to apply a NamedTransform in the specified direction.

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const *ConstNamedTransformRcPtr* &namedTransform, *TransformDirection* direction) const

ConstProcessorRcPtr **getProcessor**(const char *namedTransformName, *TransformDirection* direction) const

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const char *namedTransformName, *TransformDirection* direction) const

ConstProcessorRcPtr **getProcessor**(const *ConstTransformRcPtr* &transform) const

Get the processor for the specified transform.

Not often needed, but will allow for the re-use of atomic OCIO functionality (such as to apply an individual LUT file).

ConstProcessorRcPtr **getProcessor**(const *ConstTransformRcPtr* &transform, *TransformDirection* direction) const

ConstProcessorRcPtr **getProcessor**(const *ConstContextRcPtr* &context, const *ConstTransformRcPtr* &transform, *TransformDirection* direction) const

ProcessorCacheFlags **getProcessorCacheFlags**() const noexcept

Get the Processor Cache flags.

void **setProcessorCacheFlags**(*ProcessorCacheFlags* flags) const noexcept

Control the caching of processors in the config instance. By default, caching is on.

The flags allow turning caching off entirely or only turning it off if dynamic properties are being used by the processor.

void **clearProcessorCache**() noexcept

Clears this config's cache of Processor, CPUProcessor, and GPUProcessor instances.

This must be done if any of the LUT files used by these Processors have been modified. Note that setProcessorCacheFlags(PROCESSOR_CACHE_OFF) turns off caching but does not clear any existing cache.

void **setConfigIOProxy**(ConfigIOProxyRcPtr ciop)

Set the ConfigIOProxy object used to provision the config and LUTs from somewhere other than the file system. (This is set on the config's embedded Context object.)

ConfigIOProxyRcPtr **getConfigIOProxy**() const

bool **isArchivable**() const

Verify if the config is archivable.

A config is not archivable if any of the following are true: — The working directory is not set — It contains FileTransforms with a src outside the working directory — The search path contains paths outside the working directory — The search path contains paths that start with a context variable

Context variables are allowed but the intent is that they may only resolve to paths that are within or below the working directory. This is because the archiving function will only archive files that are within the working directory in order to ensure that if it is later expanded, that it will not create any files outside this directory.

For example, a context variable on the search path intended to contain the name of a sub-directory under the working directory must have the form "./$DIR_NAME" rather than just "$DIR_NAME" to be considered archivable. This is imperfect since there is no way to prevent the context variable from creating a path outside the working dir, but it should at least draw attention to the fact that the archive would fail if used with context vars that try to abuse the intended functionality.

Returns
bool Archivable if true.

void **archive**(std::ostream &ostream) const

Archive the config and its LUTs into the specified output stream.

The config is archived by serializing the Config object into a file named "config.ocio" and then walking through the current working directory and any sub-directories. Any files that have an extension matching a supported LUT file format are added to the archive. Any files that do not have an extension (or have some unsupported LUT extension, including .ocio), will not be added to the archive. To reiterate, it is the in-memory Config object that is archived, and not any .ocio file in the current working directory. The directory structure relative to the working directory is preserved. No files outside the working directory are archived so that if it is later expanded, no files will be created outside the working dir.

The reason the archive is created using all supported LUT file extensions rather than by trying to resolve all the FileTransforms in the Config to specific files is because of the goal to allow context variables to continue to work.

If a Config is created with CreateFromStream, CreateFromFile with an OCIOZ archive, or CreateFromConfigIOProxy, it cannot be archived unless the working directory is manually set to a directory that contains any necessary LUT files.

The provided output stream must be closed by the caller, if necessary (e.g., an ofstream).

Parameters
ostream – The output stream to write to.

Config(const *Config*&) = delete

Config &**operator=**(const *Config*&) = delete

~Config()

Do not use (needed only for pybind11).

Public Static Functions

static *ConfigRcPtr* **Create**()

Create an empty config of the current version.

Note that an empty config will not pass validation since required elements will be missing.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateRaw**()

Create a fall-back config.

This may be useful to allow client apps to launch in cases when the supplied config path is not loadable.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateFromEnv**()

Create a configuration using the OCIO environment variable.

Also supports the OCIO URI format for Built-in configs and supports archived configs. See Config::CreateFromFile.

If the variable is missing or empty, returns the same result as Config::CreateRaw.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateFromFile**(const char *filename)

Create a configuration using a specific config file.

Supports the OCIO URI format for Built-in configs. See Config::CreateFromBuiltinConfig.

Supports archived configs (.ocioz files).

Throws
Exception – If the file may not be read or does not parse.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateFromStream**(std::istream &istream)

Create a configuration using a stream.

Note that CreateFromStream does not set the working directory so the caller would need to set that separately in order to resolve FileTransforms. This function is typically only used for self-contained configs (no LUTs).

Configs created from CreateFromStream can not be archived unless the working directory is set and contains any necessary LUT files.

Parameters
istream – Stream to the config.

Throws
Exception – If the stream does not parse.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateFromConfigIOProxy**(ConfigIOProxyRcPtr ciop)

Create a config from the supplied ConfigIOProxy object. This allows the calling program to directly provide the config and associated LUTs rather than reading them from the standard file system.

See the ConfigIOProxy class documentation for more info.

Parameters
ciop – ConfigIOProxy object providing access to the config's files.

Throws
Exception – If the config may not be read from the proxy, or does not parse.

Returns
The Config object.

static *ConstConfigRcPtr* **CreateFromBuiltinConfig**(const char *configName)

Create a configuration using an OCIO built-in config.

Also supports the following OCIO URI format for Built-in configs: "ocio://default" - Default Built-in config. "ocio://cg-config-latest" - Latest Built-in CG config. "ocio://studio-config-latest" - Latest Built-in Studio config. "ocio://<CONFIG NAME>" - A specific Built-in config.

The available configNames are:

ACES Studio config, contains a more complete collection of color spaces and displays: "studio-config-v1.0.0_aces-v1.3_ocio-v2.1" "studio-config-v2.1.0_aces-v1.3_ocio-v2.3" "studio-config-v2.2.0_aces-v1.3_ocio-v2.4" "studio-config-v4.0.0_aces-v2.0_ocio-v2.5"

ACES CG config, basic color spaces for computer graphics apps: "cg-config-v1.0.0_aces-v1.3_ocio-v2.1" "cg-config-v2.1.0_aces-v1.3_ocio-v2.3" "cg-config-v2.2.0_aces-v1.3_ocio-v2.4" "cg-config-v4.0.0_aces-v2.0_ocio-v2.5"

More information is available at: https://github.com/AcademySoftwareFoundation/OpenColorIO-Config-ACES

Information about the available configs is available from the BuiltinConfigRegistry.

Parameters
configName – Built-in config name (with or without the "ocio://" URI prefix).

Throws
Exception – If the configName is not recognized.

Returns
One of the configs built into the OCIO library.

static char **GetDefaultFamilySeparator**() noexcept

Get the default family separator i.e. '/' .

static const char ***IdentifyBuiltinColorSpace**(const *ConstConfigRcPtr* &srcConfig, const *ConstConfigRcPtr* &builtinConfig, const char *builtinColorSpaceName)

Find the name of the color space in the source config that is the same as a color space in the default built-in config. For example, setting the builtinColorSpaceName to "sRGB - Texture" (a color space name from that config), would return the name for the corresponding sRGB texture space in the current config (or empty if it was not found). Note that this method relies on heuristics which may evolve over time and which may not work on all configs.

The method only looks at active color spaces. If the interchange roles are missing and heuristics are used, only scene-referred color spaces are searched.

Parameters

- **srcConfig** – The config to search for the desired color space.
- **builtinConfig** – The built-in config to use. See Config::CreateFromBuiltinConfig.
- **builtinColorSpaceName** – Color space name in the built-in default config.

Throws

Exception – if an interchange space cannot be found or the equivalent space cannot be found.

Returns

Matching color space name from the source config. Empty if not found.

static void **IdentifyInterchangeSpace**(const char **srcInterchangeName, const char **builtinInterchangeName, const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const *ConstConfigRcPtr* &builtinConfig, const char *builtinColorSpaceName)

Identify the two names of a common color space that exists in both the given config and the provided built-in config that may be used for converting color spaces between the two configs. If both configs have the interchange role set, than the color spaces set to that role will be returned. Otherwise, heuristics will be used to try and identify a known color space in the source config. These are the same heuristics that are used for other methods such as identifyBuiltinColorSpace and GetProcessorTo/FromBuiltinColorSpace.

Using this method in connection with GetProcessorFromConfigs is more efficient if you need to call GetProcessorTo/FromBuiltinColorSpace multiple times since it is only necessary to run the heuristics once (to identify the interchange spaces).

The srcColorSpaceName and builtinColorSpace name are used to decide which interchange role to use (scene- or display-referred). However, they are not used if the interchange roles are not present and the heuristics are used. It is actually only the ReferenceSpaceType of the color spaces that are used, so it is not necessary to call this function multiple times if all the spaces are of the same type. (These are the same arguments that would also be set if you were instead calling GetProcessorTo/FromBuiltinColorSpace.)

Parameters

- **srcInterchangeName** – **[out]** Color space name from the source config.
- **builtinInterchangeName** – **[out]** Corresponding color space name from the built-in config.
- **srcConfig** – The config to search for the desired color space.
- **srcColorSpaceName** – Color space name in the given config to convert to/from.
- **builtinConfig** – The built-in config to use. See Config::CreateFromBuiltinConfig.
- **builtinColorSpaceName** – Color space name in the default built-in config.

Throws

Exception – if either the srcInterchange or builtinInterchange cannot be identified.

static bool **AreViewsEqual**(const *ConstConfigRcPtr* &first, const *ConstConfigRcPtr* &second, const char *dispName, const char *viewName)

Compare views in a pair of configs.

Will return false if either of the views does not exist. This will return true even if the view is display-defined in one config and a reference to a shared view in the other config (both within the same display), as long as the contents match. The description text (if any) is ignored, since it is not a functional difference.

Note that the comparison is only on the strings contained in the view definition, the function does not attempt to compare that the color spaces or view transforms being referenced are identical (only that they have the same name).

static bool **AreVirtualViewsEqual**(const *ConstConfigRcPtr* &first, const *ConstConfigRcPtr* &second, const char *viewName)

Compare virtual views in a pair of configs.

Will return false if either of the virtual views does not exist. This will return true even if the virtual view is display-defined in one config and a reference to a shared virtual view in the other config, as long as the contents match.

The description text (if any) is ignored, since it is not a functional difference.

Note that the comparison is only on the strings contained in the view definition, the function does not attempt to compare that the color spaces or view transforms being referenced are identical (only that they have the same name).

static *ConstProcessorRcPtr* **GetProcessorToBuiltinColorSpace**(*ConstConfigRcPtr* srcConfig, const char *srcColorSpaceName, const char *builtinColorSpaceName)

Get a Processor to or from a known external color space.

These methods provide a way to interface color spaces in a config with known standard external color spaces. The set of external color space are those contained in the current default Built-in config. This includes common spaces such as "Linear Rec.709 (sRGB)", "sRGB - Texture", "ACEScg", and "ACES2065-1".

If the source config defines the necessary Interchange Role (typically "aces_interchange"), then the conversion will be well-defined and equivalent to calling GetProcessorFromConfigs with the source config and the Built-in config.

However, if the Interchange Roles are not present, heuristics will be used to try and identify a common color space in the source config that may be used to allow the conversion to proceed. If the heuristics fail to find a suitable space, an exception is thrown. The heuristics may evolve, so the results returned by this function for a given source config and color space may change in future releases of the library. However, the Interchange Roles are required in config versions 2.2 and higher, so it is hoped that the need for the heuristics will decrease over time.

Parameters

- **srcConfig** – The user's source config.
- **srcColorSpaceName** – The name of the color space in the source config.
- **builtinColorSpaceName** – The name of the color space in the current default Built-in config.

Throws

Exception – if either the src or builtin interchange space cannot be identified.

static *ConstProcessorRcPtr* **GetProcessorFromBuiltinColorSpace**(const char *builtinColorSpaceName, *ConstConfigRcPtr* srcConfig, const char *srcColorSpaceName)

See description of GetProcessorToBuiltinColorSpace.

Parameters

- **builtinColorSpaceName** – The name of the color space in the current default Built-in config.
- **srcConfig** – The user's source config.
- **srcColorSpaceName** – The name of the color space in the source config.

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const *ConstConfigRcPtr* &dstConfig, const char *dstColorSpaceName)

Get a processor to convert between color spaces in two separate configs.

This relies on both configs having the aces_interchange role (when srcName is scene-referred) or the role cie_xyz_d65_interchange (when srcName is display-referred) defined. An exception is thrown if that is not the case.

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstContextRcPtr* &srcContext, const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const *ConstContextRcPtr* &dstContext, const *ConstConfigRcPtr* &dstConfig, const char *dstColorSpaceName)

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const char *srcInterchangeName, const *ConstConfigRcPtr* &dstConfig, const char *dstColorSpaceName, const char *dstInterchangeName)

The srcInterchangeName and dstInterchangeName must refer to a pair of color spaces in the two configs that are the same. A role name may also be used.

Note: For all of the two-config GetProcessor functions, if either the source or destination color spaces are data spaces, the entire processor will be a no-op.

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstContextRcPtr* &srcContext, const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const char *srcInterchangeName, const *ConstContextRcPtr* &dstContext, const *ConstConfigRcPtr* &dstConfig, const char *dstColorSpaceName, const char *dstInterchangeName)

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const *ConstConfigRcPtr* &dstConfig, const char *dstDisplay, const char *dstView, *TransformDirection* direction)

Get a processor to convert from a color space to a display and view in two separate configs.

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstContextRcPtr* &srcContext, const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const *ConstContextRcPtr* &dstContext, const *ConstConfigRcPtr* &dstConfig, const char *dstDisplay, const char *dstView, *TransformDirection* direction)

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const char *srcInterchangeName, const *ConstConfigRcPtr* &dstConfig, const char *dstDisplay, const char *dstView, const char *dstInterchangeName, *TransformDirection* direction)

The srcInterchangeName and dstInterchangeName must refer to a pair of color spaces in the two configs that are the same. A role name may also be used.

static *ConstProcessorRcPtr* **GetProcessorFromConfigs**(const *ConstContextRcPtr* &srcContext, const *ConstConfigRcPtr* &srcConfig, const char *srcColorSpaceName, const char *srcInterchangeName, const *ConstContextRcPtr* &dstContext, const *ConstConfigRcPtr* &dstConfig, const char *dstDisplay, const char *dstView, const char *dstInterchangeName, *TransformDirection* direction)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *Config*&)

typedef std::shared_ptr<const *Config*> OpenColorIO_v2_5::**ConstConfigRcPtr**

typedef std::shared_ptr<*Config*> OpenColorIO_v2_5::**ConfigRcPtr**

Constants: *Roles*, *Shared View*

5.7.6 Constants

Environment Variables

These environmental variables are used by the OpenColorIO library (i.e. these variables are defined in src/OpenColorIO/Config.cpp).

Python

PyOpenColorIO.**OCIO_CONFIG_ENVVAR**

The envvar 'OCIO' provides a path to the config file used by Config.CreateFromEnv

PyOpenColorIO.**OCIO_ACTIVE_DISPLAYS_ENVVAR**

The envvar 'OCIO_ACTIVE_DISPLAYS' provides a list of displays overriding the 'active_displays' list from the config file.

PyOpenColorIO.**OCIO_ACTIVE_VIEWS_ENVVAR**

The envvar 'OCIO_ACTIVE_VIEWS' provides a list of views overriding the 'active_views' list from the config file.

PyOpenColorIO.**OCIO_INACTIVE_COLORSPACES_ENVVAR**

The envvar 'OCIO_INACTIVE_COLORSPACES' provides a list of inactive color spaces overriding the 'inactive_color_spaces' list from the config file.

PyOpenColorIO.**OCIO_OPTIMIZATION_FLAGS_ENVVAR**

The envvar 'OCIO_OPTIMIZATION_FLAGS' provides a way to force a given optimization level. Remove the variable or set the value to empty to not use it. Set the value of the variable to the desired optimization level as either an integer or hexadecimal value. Ex: OCIO_OPTIMIZATION_FLAGS="20479" or "0x4FFF" for OPTIMIZATION_LOSSLESS.

C++

Warning: doxygengroup: Cannot find group "VarsEnvvar" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

Roles

ColorSpace Roles are used so that plugins, in addition to this API can have abstract ways of asking for common colorspaces, without referring to them by hardcoded names.

Internal:

```
Extracting color space from file path - (ROLE_DEFAULT)
```

App Helpers:

```
ViewingPipeline          - (ROLE_SCENE_LINEAR (LinearCC for exposure))
                           (ROLE_COLOR_TIMING (ColorTimingCC))
MixingColorSpaceManager  - (ROLE_COLOR_PICKING)
```

External Plugins (currently known):

```
Colorpicker UIs       - (ROLE_COLOR_PICKING)
Compositor LogConvert - (ROLE_SCENE_LINEAR, ROLE_COMPOSITING_LOG)
```

Python

PyOpenColorIO.**ROLE_DEFAULT**

"default"

PyOpenColorIO.**ROLE_REFERENCE**

"reference"

PyOpenColorIO.**ROLE_DATA**

"data"

PyOpenColorIO.**ROLE_COLOR_PICKING**

"color_picking"

PyOpenColorIO.**ROLE_SCENE_LINEAR**

"scene_linear"

PyOpenColorIO.**ROLE_COMPOSITING_LOG**

"compositing_log"

PyOpenColorIO.**ROLE_COLOR_TIMING**

"color_timing"

PyOpenColorIO.**ROLE_TEXTURE_PAINT**

This role defines the transform for painting textures. In some workflows this is just a inverse display gamma with some limits.

PyOpenColorIO.**ROLE_MATTE_PAINT**

This role defines the transform for matte painting. In some workflows this is a 1D HDR to LDR allocation. It is normally combined with another display transform in the host app for preview.

PyOpenColorIO.**ROLE_RENDERING**

The rendering role may be used to identify a specific color space to be used by CGI renderers. This is typically a scene-linear space but the primaries also matter since they influence the resulting color, especially in areas of indirect illumination.

PyOpenColorIO.**ROLE_INTERCHANGE_SCENE**

The aces_interchange role is used to specify which color space in the config implements the standard ACES2065-1 color space (SMPTE ST2065-1). This may be used when converting scene-referred colors from one config to another.

PyOpenColorIO.**ROLE_INTERCHANGE_DISPLAY**

The cie_xyz_d65_interchange role is used to specify which color space in the config implements CIE XYZ colorimetry with the neutral axis at D65. This may be used when converting display-referred colors from one config to another.

C++

Warning: doxygengroup: Cannot find group "VarsRoles" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

Shared View

Python

PyOpenColorIO.**OCIO_VIEW_USE_DISPLAY_NAME**

A shared view using this for the color space name will use a display color space that has the same name as the display the shared view is used by.

C++

Warning: doxygengroup: Cannot find group "VarsSharedView" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

FormatMetadata

These constants describe various types of rich metadata. They are used with FormatMetadata objects as the "name" part of a (name, value) pair. All of these types of metadata are supported in the CLF/CTF file formats whereas other formats support some or none of them.

Although the string constants used here match those used in the CLF/CTF formats, the concepts are generic, so the goal is for other file formats to reuse the same constants within a FormatMetadata object (even if the syntax used in a given format is somewhat different).

Python

PyOpenColorIO.**METADATA_DESCRIPTION**

A description string – used as the "Description" element in CLF/CTF and CDL, and to hold comments for other LUT formats when baking.

PyOpenColorIO.**METADATA_INFO**

A block of informative metadata such as the "Info" element in CLF/CTF. Usually contains child elements.

PyOpenColorIO.**METADATA_INPUT_DESCRIPTOR**

A string describing the expected input color space – used as the "InputDescriptor" element in CLF/CTF and the "InputDescription" in CDL.

PyOpenColorIO.**METADATA_OUTPUT_DESCRIPTOR**

A string describing the output color space – used as the "OutputDescriptor" element in CLF/CTF and the "OutputDescription" in CDL.

PyOpenColorIO.**METADATA_NAME**

A name string – used as a "name" attribute in CLF/CTF elements. Use on a GroupTransform to get/set the name for the CLF/CTF ProcessList. Use on an individual Transform (i.e. MatrixTransform, etc.) to get/set the name of the corresponding process node.

PyOpenColorIO.**METADATA_ID**

An ID string – used as an "id" attribute in CLF/CTF elements. Use on a GroupTransform to get/set the id for the CLF/CTF ProcessList. Use on an individual Transform (i.e. MatrixTransform, etc.) to get/set the id of the corresponding process node.

C++

Warning: doxygengroup: Cannot find group "VarsFormatMetadata" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

Caches

Python

PyOpenColorIO.**OCIO_DISABLE_ALL_CACHES**

Disable all caches, including for FileTransforms and Optimized/CPU/GPU Processors. (Provided only to facilitate developer investigations.)

PyOpenColorIO.**OCIO_DISABLE_PROCESSOR_CACHES**

Disable only the Optimized, CPU, and GPU Processor caches. (Provided only to facilitate developer investigations.)

PyOpenColorIO.**OCIO_DISABLE_CACHE_FALLBACK**

By default the processor caches check for identical color transformations when cache keys do not match. That fallback introduces a major performance hit in some cases so there is an env. variable to disable the fallback.

C++

Warning: doxygengroup: Cannot find group "VarsCaches" in doxygen xml output for project "OpenColorIO" from directory: _doxygen/xml

5.7.7 Context

Python

class PyOpenColorIO.**Context**

Context

A context defines some overrides to a Config. For example, it can override the search path or change the value of a context variable.

Note: Only some Config::getProcessor methods accept a custom context; otherwise, the default context instance is used (see Config::getCurrentContext).

Context Variables

The context variables allow changes at runtime using environment variables. For example, a color space name (such as src & dst for the ColorSpaceTransform) or a file name (such as LUT file name for the FileTransform) could be defined by context variables. The color transformation is then customized based on some environment variables.

In a config the context variables support three syntaxes (i.e. ${VAR}, $VAR and VAR%) and the parsing starts from longest to shortest. So, the resolve works like '$TEST_$TESTING_$TE' expands in this order '2 1 3'.

Config authors are recommended to include the "environment" section in their configs. This improves performance as well as making the config more readable. When present, this section must declare all context variables used in the config. It may also provide a default value, in case the variable is not present in the user's environment.

A context variable may only be used in the following places: - the *:ref: `ColorSpaceTransform `* to define the source and the destination color space names, - the *:ref: `FileTransform `* to define the source file name (e.g. a LUT file name), - the search_path, - the cccid of the *:ref: `FileTransform `* to only extract one specific transform from the CDL & CCC files.

Some specific restrictions are worth calling out: - they cannot be used as either the name or value of a role, - the context variable characters $ and % are prohibited in a color space name.

Context(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.Context) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.Context, workingDir: str = '', searchPaths: collections.abc.Sequence[str] = [], stringVars: collections.abc.Mapping[str, str] = {}, environmentMode: PyOpenColorIO.PyOpenColorIO.EnvironmentMode = <EnvironmentMode.ENV_ENVIRONMENT_LOAD_PREDEFINED: 1>) -> None

addSearchPath(*path: str*) → None

clearSearchPaths() → None

clearStringVars() → None

getCacheID() → str

getEnvironmentMode() → *PyOpenColorIO.PyOpenColorIO.EnvironmentMode*

getSearchPath() → str

getSearchPaths() → *PyOpenColorIO.PyOpenColorIO.Context.SearchPathIterator*

getStringVars() → *PyOpenColorIO.PyOpenColorIO.Context.StringVarIterator*

getWorkingDir() → str

loadEnvironment() → None

Seed string vars with the current environment, based on the EnvironmentMode setting.

resolveFileLocation(**args*, ***kwargs*)

Overloaded function.

1. resolveFileLocation(self: PyOpenColorIO.PyOpenColorIO.Context, filename: str) -> str

Build the resolved and expanded filepath using the search_path when needed, and check if the filepath exists. If it cannot be resolved or found, an exception will be thrown. The method argument is directly from the config file so it can be an absolute or relative file path or a file name.

Note: The filepath existence check could add a performance hit.

Note: The context variable resolution is performed using `resolveStringVar()`.

2. resolveFileLocation(self: PyOpenColorIO.PyOpenColorIO.Context, filename: str, usedContextVars: PyOpenColorIO.PyOpenColorIO.Context) -> str

Build the resolved and expanded filepath and return all the context variables used to resolve the filename (empty if no context variables were used).

resolveStringVar(**args*, ***kwargs*)

Overloaded function.

1. resolveStringVar(self: PyOpenColorIO.PyOpenColorIO.Context, string: str) -> str

Resolve all the context variables from the string. It could be color space names or file names. Note that it recursively applies the context variable resolution. Returns the string unchanged if it does not contain any context variable.

2. resolveStringVar(self: PyOpenColorIO.PyOpenColorIO.Context, string: str, usedContextVars: PyOpenColorIO.PyOpenColorIO.Context) -> str

Resolve all the context variables from the string and return all the context variables used to resolve the string (empty if no context variables were used).

setEnvironmentMode(*mode:* PyOpenColorIO.PyOpenColorIO.EnvironmentMode) → None

See Config::setEnvironmentMode.

setSearchPath(*path: str*) → None

setStringVar(*name: str*, *value: str*) → None

Add (or update) a context variable. But it removes it if the value argument is null. Note that a Context StringVar is the same thing as a Config EnvironmentVar and these are both often referred to as a "context var".

setWorkingDir(*dirName: str*) → None

class PyOpenColorIO.Context.**StringVarNameIterator**

self **[***arg0: SupportsInt***]** → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Context.StringVarNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.Context.**StringVarIterator**

self **[***arg0: SupportsInt***]** → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Context.StringVarIterator*

len(*self*) → int

next(*self*) → tuple

class PyOpenColorIO.Context.**SearchPathIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Context.SearchPathIterator*

len(*self*) → int

next(*self*) → str

C++

class **Context**

Context

A context defines some overrides to a Config. For example, it can override the search path or change the value of a context variable.

Context Variables

The context variables allow changes at runtime using environment variables. For example, a color space name (such as src & dst for the ColorSpaceTransform) or a file name (such as LUT file name for the FileTransform) could be defined by context variables. The color transformation is then customized based on some environment variables.

In a config the context variables support three syntaxes (i.e. ${VAR}, $VAR and VAR%) and the parsing starts from longest to shortest. So, the resolve works like '$TEST_$TESTING_$TE' expands in this order '2 1 3'.

Config authors are recommended to include the "environment" section in their configs. This improves performance as well as making the config more readable. When present, this section must declare all context variables used in the config. It may also provide a default value, in case the variable is not present in the user's environment.

A context variable may only be used in the following places:

- the `ColorSpaceTransform` to define the source and the destination color space names,
- the `FileTransform` to define the source file name (e.g. a LUT file name),
- the search_path,
- the cccid of the `FileTransform` to only extract one specific transform from the CDL & CCC files.

Some specific restrictions are worth calling out:

- they cannot be used as either the name or value of a role,
- the context variable characters $ and % are prohibited in a color space name.

Note: Only some Config::getProcessor methods accept a custom context; otherwise, the default context instance is used (see Config::getCurrentContext).

Public Functions

ContextRcPtr **createEditableCopy**() const

const char ***getCacheID**() const

void **setSearchPath**(const char *path)

const char ***getSearchPath**() const

int **getNumSearchPaths**() const

const char ***getSearchPath**(int index) const

void **clearSearchPaths**()

void **addSearchPath**(const char *path)

void **setWorkingDir**(const char *dirname)

const char ***getWorkingDir**() const

void **setStringVar**(const char *name, const char *value) noexcept

Add (or update) a context variable. But it removes it if the value argument is null. Note that a Context StringVar is the same thing as a Config EnvironmentVar and these are both often referred to as a "context var".

const char ***getStringVar**(const char *name) const noexcept

Get the context variable value. It returns an empty string if the context variable is null or does not exist.

int **getNumStringVars**() const

const char ***getStringVarNameByIndex**(int index) const

const char ***getStringVarByIndex**(int index) const

void **clearStringVars**()

void **addStringVars**(const *ConstContextRcPtr* &ctx) noexcept

Add to the instance all the context variables from ctx.

void **setEnvironmentMode**(*EnvironmentMode* mode) noexcept

See Config::setEnvironmentMode.

EnvironmentMode **getEnvironmentMode**() const noexcept

void **loadEnvironment**() noexcept

Seed string vars with the current environment, based on the EnvironmentMode setting.

const char ***resolveStringVar**(const char *string) const noexcept

Resolve all the context variables from the string. It could be color space names or file names. Note that it recursively applies the context variable resolution. Returns the string unchanged if it does not contain any context variable.

const char ***resolveStringVar**(const char *string, *ContextRcPtr* &usedContextVars) const noexcept

Resolve all the context variables from the string and return all the context variables used to resolve the string (empty if no context variables were used).

const char *resolveFileLocation(const char *filename) const

Build the resolved and expanded filepath using the search_path when needed, and check if the filepath exists. If it cannot be resolved or found, an exception will be thrown. The method argument is directly from the config file so it can be an absolute or relative file path or a file name.

Note: The filepath existence check could add a performance hit.

Note: The context variable resolution is performed using :cpp:func:`resolveStringVar`.

const char *resolveFileLocation(const char *filename, *ContextRcPtr* &usedContextVars) const

Build the resolved and expanded filepath and return all the context variables used to resolve the filename (empty if no context variables were used).

void setConfigIOProxy(ConfigIOProxyRcPtr ciop)

Set the ConfigIOProxy object used to provision the config and LUTs from somewhere other than the file system.

ConfigIOProxyRcPtr getConfigIOProxy() const

Context(const *Context*&) = delete

Context &operator=(const *Context*&) = delete

~Context()

Do not use (needed only for pybind11).

Public Static Functions

static *ContextRcPtr* Create()

std::ostream &OpenColorIO_v2_5::operator<<(std::ostream&, const *Context*&)

typedef std::shared_ptr<const *Context*> OpenColorIO_v2_5::ConstContextRcPtr

typedef std::shared_ptr<*Context*> OpenColorIO_v2_5::ContextRcPtr

5.7.8 DynamicProperty

Python

class PyOpenColorIO.DynamicProperty

Allows transform parameter values to be set on-the-fly (after finalization). For example, to modify the exposure in a viewport. Dynamic properties can be accessed from the *:ref: `CPUProcessor `* or *:ref: `GpuShaderCreator `* to change values between processing.

```
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();
OCIO::ConstProcessorRcPtr processor = config->getProcessor(colorSpace1,␣
↪colorSpace2);
OCIO::ConstCPUProcessorRcPtr cpuProcessor = processor->getDefaultCPUProcessor();
```

(continues on next page)

(continued from previous page)

```
if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_EXPOSURE))
{
    // Get the in-memory implementation of the dynamic property.
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_EXPOSURE);
    // Get the interface used to change the double value.
    OCIO::DynamicPropertyDoubleRcPtr exposure =
        OCIO::DynamicPropertyValue::AsDouble(dynProp);
    // Update of the dynamic property instance with the new value.
    exposure->setValue(1.1f);
}
if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_PRIMARY))
{
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_PRIMARY);
    OCIO::DynamicPropertyGradingPrimaryRcPtr primaryProp =
        OCIO::DynamicPropertyValue::AsGradingPrimary(dynProp);
    OCIO::GradingPrimary primary = primaryProp->getValue();
    primary.m_saturation += 0.1f;
    primaryProp->setValue(primary);
}
if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_RGBCURVE))
{
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_RGBCURVE);
    OCIO::DynamicPropertyGradingRGBCurveRcPtr rgbCurveProp =
        OCIO::DynamicPropertyValue::AsGradingRGBCurve(dynProp);
    OCIO::ConstGradingRGBCurveRcPtr rgbCurve = rgbCurveProp->getValue()->
↪createEditableCopy();
    OCIO::GradingBSplineCurveRcPtr rCurve = rgbCurve->getCurve(OCIO::RGB_RED);
    rCurve->getControlPoint(1).m_y += 0.1f;
    rgbCurveProp->setValue(rgbCurve);
}
```

DynamicProperty**args*, ***kwargs*

getDouble() → float

Get the property as DynamicPropertyDoubleRcPtr to access the double value. Will throw if property type is not a type that holds a double such as DYNAMIC_PROPERTY_EXPOSURE.

getGradingHueCurve() → *PyOpenColorIO.PyOpenColorIO.GradingHueCurve*

Get the property as DynamicPropertyGradingHueCurveRcPtr to access the GradingHueCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_HUECURVE.

getGradingPrimary() → *PyOpenColorIO.PyOpenColorIO.GradingPrimary*

Get the property as DynamicPropertyGradingPrimaryRcPtr to access the GradingPrimary value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_PRIMARY.

getGradingRGBCurve() → *PyOpenColorIO.PyOpenColorIO.GradingRGBCurve*

Get the property as DynamicPropertyGradingRGBCurveRcPtr to access the GradingRGBCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_RGBCURVE.

getGradingTone() → *PyOpenColorIO.PyOpenColorIO.GradingTone*

Get the property as DynamicPropertyGradingToneRcPtr to access the GradingTone value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_TONE.

getType() → *PyOpenColorIO.PyOpenColorIO.DynamicPropertyType*

setDouble(*val: SupportsFloat*) → None

Get the property as DynamicPropertyDoubleRcPtr to access the double value. Will throw if property type is not a type that holds a double such as DYNAMIC_PROPERTY_EXPOSURE.

setGradingHueCurve(*val:* PyOpenColorIO.PyOpenColorIO.GradingHueCurve) → None

Get the property as DynamicPropertyGradingHueCurveRcPtr to access the GradingHueCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_HUECURVE.

setGradingPrimary(*val:* PyOpenColorIO.PyOpenColorIO.GradingPrimary) → None

Get the property as DynamicPropertyGradingPrimaryRcPtr to access the GradingPrimary value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_PRIMARY.

setGradingRGBCurve(*val:* PyOpenColorIO.PyOpenColorIO.GradingRGBCurve) → None

Get the property as DynamicPropertyGradingRGBCurveRcPtr to access the GradingRGBCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_RGBCURVE.

setGradingTone(*val:* PyOpenColorIO.PyOpenColorIO.GradingTone) → None

Get the property as DynamicPropertyGradingToneRcPtr to access the GradingTone value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_TONE.

C++

class **DynamicProperty**

Allows transform parameter values to be set on-the-fly (after finalization). For example, to modify the exposure in a viewport. Dynamic properties can be accessed from the `CPUProcessor` or `GpuShaderCreator` to change values between processing.

```
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();
OCIO::ConstProcessorRcPtr processor = config->getProcessor(colorSpace1,␣
↪colorSpace2);
OCIO::ConstCPUProcessorRcPtr cpuProcessor = processor->getDefaultCPUProcessor();

if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_EXPOSURE))
{
    // Get the in-memory implementation of the dynamic property.
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_EXPOSURE);
    // Get the interface used to change the double value.
    OCIO::DynamicPropertyDoubleRcPtr exposure =
        OCIO::DynamicPropertyValue::AsDouble(dynProp);
    // Update of the dynamic property instance with the new value.
    exposure->setValue(1.1f);
}
if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_PRIMARY))
{
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_PRIMARY);
    OCIO::DynamicPropertyGradingPrimaryRcPtr primaryProp =
        OCIO::DynamicPropertyValue::AsGradingPrimary(dynProp);
```

(continues on next page)

(continued from previous page)

```
    OCIO::GradingPrimary primary = primaryProp->getValue();
    primary.m_saturation += 0.1f;
    primaryProp->setValue(primary);
}
if (cpuProcessor->hasDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_RGBCURVE))
{
    OCIO::DynamicPropertyRcPtr dynProp =
        cpuProcessor->getDynamicProperty(OCIO::DYNAMIC_PROPERTY_GRADING_RGBCURVE);
    OCIO::DynamicPropertyGradingRGBCurveRcPtr rgbCurveProp =
        OCIO::DynamicPropertyValue::AsGradingRGBCurve(dynProp);
    OCIO::ConstGradingRGBCurveRcPtr rgbCurve = rgbCurveProp->getValue()->
↪createEditableCopy();
    OCIO::GradingBSplineCurveRcPtr rCurve = rgbCurve->getCurve(OCIO::RGB_RED);
    rCurve->getControlPoint(1).m_y += 0.1f;
    rgbCurveProp->setValue(rgbCurve);
}
```

Public Functions

virtual *DynamicPropertyType* **getType**() const noexcept = 0

DynamicProperty &**operator=**(const *DynamicProperty*&) = delete

DynamicProperty(const *DynamicProperty*&) = delete

virtual **~DynamicProperty**() = default

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *DynamicProperty*> OpenColorIO_v2_5::**ConstDynamicPropertyRcPtr**

typedef std::shared_ptr<*DynamicProperty*> OpenColorIO_v2_5::**DynamicPropertyRcPtr**

5.7.9 DynamicPropertyValue

C++

namespace **DynamicPropertyValue**

Functions

DynamicPropertyDoubleRcPtr **AsDouble**(*DynamicPropertyRcPtr* &prop)

Get the property as DynamicPropertyDoubleRcPtr to access the double value. Will throw if property type is not a type that holds a double such as DYNAMIC_PROPERTY_EXPOSURE.

DynamicPropertyGradingPrimaryRcPtr **AsGradingPrimary**(*DynamicPropertyRcPtr* &prop)

Get the property as DynamicPropertyGradingPrimaryRcPtr to access the GradingPrimary value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_PRIMARY.

DynamicPropertyGradingRGBCurveRcPtr **AsGradingRGBCurve**(*DynamicPropertyRcPtr* &prop)

Get the property as DynamicPropertyGradingRGBCurveRcPtr to access the GradingRGBCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_RGBCURVE.

DynamicPropertyGradingHueCurveRcPtr **AsGradingHueCurve**(*DynamicPropertyRcPtr* &prop)

Get the property as DynamicPropertyGradingHueCurveRcPtr to access the GradingHueCurveRcPtr value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_HUECURVE.

DynamicPropertyGradingToneRcPtr **AsGradingTone**(*DynamicPropertyRcPtr* &prop)

Get the property as DynamicPropertyGradingToneRcPtr to access the GradingTone value. Will throw if property type is not DYNAMIC_PROPERTY_GRADING_TONE.

5.7.10 DynamicPropertyDouble

C++

class **DynamicPropertyDouble**

Interface used to access dynamic property double value.

Public Functions

virtual double **getValue**() const = 0

virtual void **setValue**(double value) = 0

DynamicPropertyDouble(const *DynamicPropertyDouble*&) = delete

DynamicPropertyDouble &**operator=**(const *DynamicPropertyDouble*&) = delete

virtual **~DynamicPropertyDouble**() = default

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *DynamicPropertyDouble*>
OpenColorIO_v2_5::**ConstDynamicPropertyDoubleRcPtr**

typedef std::shared_ptr<*DynamicPropertyDouble*> OpenColorIO_v2_5::**DynamicPropertyDoubleRcPtr**

5.7.11 DynamicPropertyGradingPrimary

C++

class **DynamicPropertyGradingPrimary**

Interface used to access dynamic property GradingPrimary value.

Public Functions

virtual const *GradingPrimary* &**getValue**() const = 0

virtual void **setValue**(const *GradingPrimary* &value) = 0

Will throw if value is not valid.

DynamicPropertyGradingPrimary(const *DynamicPropertyGradingPrimary*&) = delete

DynamicPropertyGradingPrimary &**operator=**(const *DynamicPropertyGradingPrimary*&) = delete

virtual **~DynamicPropertyGradingPrimary**() = default

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *DynamicPropertyGradingPrimary*>
OpenColorIO_v2_5::**ConstDynamicPropertyGradingPrimaryRcPtr**

typedef std::shared_ptr<*DynamicPropertyGradingPrimary*>
OpenColorIO_v2_5::**DynamicPropertyGradingPrimaryRcPtr**

5.7.12 DynamicPropertyGradingRGBCurve

C++

class **DynamicPropertyGradingRGBCurve**

Interface used to access dynamic property ConstGradingRGBCurveRcPtr value.

Public Functions

virtual const *ConstGradingRGBCurveRcPtr* &**getValue**() const = 0

virtual void **setValue**(const *ConstGradingRGBCurveRcPtr* &value) = 0

Will throw if value is not valid.

DynamicPropertyGradingRGBCurve(const *DynamicPropertyGradingRGBCurve*&) = delete

DynamicPropertyGradingRGBCurve &**operator=**(const *DynamicPropertyGradingRGBCurve*&) = delete

virtual **~DynamicPropertyGradingRGBCurve**() = default

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *DynamicPropertyGradingRGBCurve*>
OpenColorIO_v2_5::**ConstDynamicPropertyGradingRGBCurveRcPtr**

typedef std::shared_ptr<*DynamicPropertyGradingRGBCurve*>
OpenColorIO_v2_5::**DynamicPropertyGradingRGBCurveRcPtr**

5.7.13 DynamicPropertyGradingTone

C++

class **DynamicPropertyGradingTone**

Interface used to access dynamic property GradingTone value.

Public Functions

virtual const *GradingTone* &**getValue**() const = 0

virtual void **setValue**(const *GradingTone* &value) = 0

Will throw if value is not valid.

DynamicPropertyGradingTone(const *DynamicPropertyGradingTone*&) = delete

DynamicPropertyGradingTone &**operator=**(const *DynamicPropertyGradingTone*&) = delete

virtual **~DynamicPropertyGradingTone**() = default

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *DynamicPropertyGradingTone*>
OpenColorIO_v2_5::**ConstDynamicPropertyGradingToneRcPtr**

typedef std::shared_ptr<*DynamicPropertyGradingTone*>
OpenColorIO_v2_5::**DynamicPropertyGradingToneRcPtr**

5.7.14 Enums

LoggingLevel

Python

class PyOpenColorIO.**LoggingLevel**

Members:

LOGGING_LEVEL_NONE :

LOGGING_LEVEL_WARNING :

LOGGING_LEVEL_INFO :

LOGGING_LEVEL_DEBUG :

LOGGING_LEVEL_UNKNOWN :

LOGGING_LEVEL_DEBUG = <LoggingLevel.LOGGING_LEVEL_DEBUG: 3>

LOGGING_LEVEL_INFO = <LoggingLevel.LOGGING_LEVEL_INFO: 2>

LOGGING_LEVEL_NONE = <LoggingLevel.LOGGING_LEVEL_NONE: 0>

LOGGING_LEVEL_UNKNOWN = <LoggingLevel.LOGGING_LEVEL_UNKNOWN: 255>

LOGGING_LEVEL_WARNING = <LoggingLevel.LOGGING_LEVEL_WARNING: 1>

property value

C++

enum OpenColorIO_v2_5::**LoggingLevel**

Values:

enumerator **LOGGING_LEVEL_NONE**

enumerator **LOGGING_LEVEL_WARNING**

enumerator **LOGGING_LEVEL_INFO**

enumerator **LOGGING_LEVEL_DEBUG**

enumerator **LOGGING_LEVEL_UNKNOWN**

enumerator **LOGGING_LEVEL_DEFAULT**

Conversions: *LoggingLevel*

ReferenceSpaceType

Python

class PyOpenColorIO.**ReferenceSpaceType**

OCIO does not mandate the image state of the main reference space and it is not required to be scene-referred. This enum is used in connection with the display color space and view transform features which do assume that the main reference space is scene-referred and the display reference space is display-referred. If a config used a non-scene-referred reference space, presumably it would not use either display color spaces or view transforms, so this enum becomes irrelevant.

Members:

REFERENCE_SPACE_SCENE : the main scene reference space

REFERENCE_SPACE_DISPLAY : the reference space for display color spaces

REFERENCE_SPACE_DISPLAY = <ReferenceSpaceType.REFERENCE_SPACE_DISPLAY: 1>

REFERENCE_SPACE_SCENE = <ReferenceSpaceType.REFERENCE_SPACE_SCENE: 0>

property value

C++

enum OpenColorIO_v2_5::**ReferenceSpaceType**

OCIO does not mandate the image state of the main reference space and it is not required to be scene-referred. This enum is used in connection with the display color space and view transform features which do assume that the main reference space is scene-referred and the display reference space is display-referred. If a config used a non-scene-referred reference space, presumably it would not use either display color spaces or view transforms, so this enum becomes irrelevant.

Values:

enumerator **REFERENCE_SPACE_SCENE**

the main scene reference space

enumerator **REFERENCE_SPACE_DISPLAY**

the reference space for display color spaces

SearchReferenceSpaceType

Python

class PyOpenColorIO.**SearchReferenceSpaceType**

Members:

SEARCH_REFERENCE_SPACE_SCENE :

SEARCH_REFERENCE_SPACE_DISPLAY :

SEARCH_REFERENCE_SPACE_ALL :

SEARCH_REFERENCE_SPACE_ALL = <SearchReferenceSpaceType.SEARCH_REFERENCE_SPACE_ALL: 2>

SEARCH_REFERENCE_SPACE_DISPLAY = <SearchReferenceSpaceType.SEARCH_REFERENCE_SPACE_DISPLAY: 1>

SEARCH_REFERENCE_SPACE_SCENE = <SearchReferenceSpaceType.SEARCH_REFERENCE_SPACE_SCENE: 0>

property value

C++

enum OpenColorIO_v2_5::**SearchReferenceSpaceType**

Values:

enumerator **SEARCH_REFERENCE_SPACE_SCENE**

enumerator **SEARCH_REFERENCE_SPACE_DISPLAY**

enumerator **SEARCH_REFERENCE_SPACE_ALL**

NamedTransformVisibility

Python

class PyOpenColorIO.**NamedTransformVisibility**

Members:

NAMEDTRANSFORM_ACTIVE :

NAMEDTRANSFORM_INACTIVE :

NAMEDTRANSFORM_ALL :

NAMEDTRANSFORM_ACTIVE = <NamedTransformVisibility.NAMEDTRANSFORM_ACTIVE: 0>

NAMEDTRANSFORM_ALL = <NamedTransformVisibility.NAMEDTRANSFORM_ALL: 2>

NAMEDTRANSFORM_INACTIVE = <NamedTransformVisibility.NAMEDTRANSFORM_INACTIVE: 1>

property value

C++

enum OpenColorIO_v2_5::**NamedTransformVisibility**

Values:

enumerator **NAMEDTRANSFORM_ACTIVE**

enumerator **NAMEDTRANSFORM_INACTIVE**

enumerator **NAMEDTRANSFORM_ALL**

ColorSpaceVisibility

Python

class PyOpenColorIO.**ColorSpaceVisibility**

Members:

COLORSPACE_ACTIVE :

COLORSPACE_INACTIVE :

COLORSPACE_ALL :

COLORSPACE_ACTIVE = <ColorSpaceVisibility.COLORSPACE_ACTIVE: 0>

COLORSPACE_ALL = <ColorSpaceVisibility.COLORSPACE_ALL: 2>

COLORSPACE_INACTIVE = <ColorSpaceVisibility.COLORSPACE_INACTIVE: 1>

property value

C++

enum OpenColorIO_v2_5::**ColorSpaceVisibility**

Values:

enumerator **COLORSPACE_ACTIVE**

enumerator **COLORSPACE_INACTIVE**

enumerator **COLORSPACE_ALL**

ViewType

Python

class PyOpenColorIO.**ViewType**

Members:

VIEW_SHARED :

VIEW_DISPLAY_DEFINED :

VIEW_DISPLAY_DEFINED = <ViewType.VIEW_DISPLAY_DEFINED: 1>

VIEW_SHARED = <ViewType.VIEW_SHARED: 0>

property value

C++

enum OpenColorIO_v2_5::**ViewType**

Values:

enumerator **VIEW_SHARED**

enumerator **VIEW_DISPLAY_DEFINED**

ColorSpaceDirection

Python

class PyOpenColorIO.**ColorSpaceDirection**

Members:

COLORSPACE_DIR_TO_REFERENCE :

COLORSPACE_DIR_FROM_REFERENCE :

COLORSPACE_DIR_FROM_REFERENCE = <ColorSpaceDirection.COLORSPACE_DIR_FROM_REFERENCE: 1>

COLORSPACE_DIR_TO_REFERENCE = <ColorSpaceDirection.COLORSPACE_DIR_TO_REFERENCE: 0>

property value

C++

enum OpenColorIO_v2_5::**ColorSpaceDirection**

Values:

enumerator **COLORSPACE_DIR_TO_REFERENCE**

enumerator **COLORSPACE_DIR_FROM_REFERENCE**

Conversions: conversion_color_space_direction

ViewTransformDirection

Python

class PyOpenColorIO.**ViewTransformDirection**

Members:

VIEWTRANSFORM_DIR_TO_REFERENCE :

VIEWTRANSFORM_DIR_FROM_REFERENCE :

VIEWTRANSFORM_DIR_FROM_REFERENCE = <ViewTransformDirection.VIEWTRANSFORM_DIR_FROM_REFERENCE: 1>

VIEWTRANSFORM_DIR_TO_REFERENCE = <ViewTransformDirection.VIEWTRANSFORM_DIR_TO_REFERENCE: 0>

property value

C++

enum OpenColorIO_v2_5::**ViewTransformDirection**

Values:

enumerator **VIEWTRANSFORM_DIR_TO_REFERENCE**

enumerator **VIEWTRANSFORM_DIR_FROM_REFERENCE**

TransformDirection

Python

class PyOpenColorIO.**TransformDirection**

Members:

TRANSFORM_DIR_FORWARD :

TRANSFORM_DIR_INVERSE :

TRANSFORM_DIR_FORWARD = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>

TRANSFORM_DIR_INVERSE = <TransformDirection.TRANSFORM_DIR_INVERSE: 1>

property value

C++

enum OpenColorIO_v2_5::**TransformDirection**

Values:

enumerator **TRANSFORM_DIR_FORWARD**

enumerator **TRANSFORM_DIR_INVERSE**

Conversions: *TransformDirection*

TransformType

Python

class PyOpenColorIO.**TransformType**

Members:

TRANSFORM_TYPE_ALLOCATION :

TRANSFORM_TYPE_BUILTIN :

TRANSFORM_TYPE_CDL :

TRANSFORM_TYPE_COLORSPACE :

TRANSFORM_TYPE_DISPLAY_VIEW :

TRANSFORM_TYPE_EXPONENT :

TRANSFORM_TYPE_EXPONENT_WITH_LINEAR :

TRANSFORM_TYPE_EXPOSURE_CONTRAST :

TRANSFORM_TYPE_FILE :

TRANSFORM_TYPE_FIXED_FUNCTION :

TRANSFORM_TYPE_GRADING_PRIMARY :

TRANSFORM_TYPE_GRADING_RGB_CURVE :

TRANSFORM_TYPE_GRADING_HUE_CURVE :

TRANSFORM_TYPE_GRADING_TONE :

TRANSFORM_TYPE_GROUP :

TRANSFORM_TYPE_LOG_AFFINE :

TRANSFORM_TYPE_LOG_CAMERA :

TRANSFORM_TYPE_LOG :

TRANSFORM_TYPE_LOOK :

TRANSFORM_TYPE_LUT1D :

TRANSFORM_TYPE_LUT3D :

TRANSFORM_TYPE_MATRIX :

TRANSFORM_TYPE_RANGE :

TRANSFORM_TYPE_ALLOCATION = <TransformType.TRANSFORM_TYPE_ALLOCATION: 0>

TRANSFORM_TYPE_BUILTIN = <TransformType.TRANSFORM_TYPE_BUILTIN: 1>

TRANSFORM_TYPE_CDL = <TransformType.TRANSFORM_TYPE_CDL: 2>

TRANSFORM_TYPE_COLORSPACE = <TransformType.TRANSFORM_TYPE_COLORSPACE: 3>

TRANSFORM_TYPE_DISPLAY_VIEW = <TransformType.TRANSFORM_TYPE_DISPLAY_VIEW: 4>

TRANSFORM_TYPE_EXPONENT = <TransformType.TRANSFORM_TYPE_EXPONENT: 5>

TRANSFORM_TYPE_EXPONENT_WITH_LINEAR =
<TransformType.TRANSFORM_TYPE_EXPONENT_WITH_LINEAR: 6>

TRANSFORM_TYPE_EXPOSURE_CONTRAST = <TransformType.TRANSFORM_TYPE_EXPOSURE_CONTRAST: 7>

TRANSFORM_TYPE_FILE = <TransformType.TRANSFORM_TYPE_FILE: 8>

TRANSFORM_TYPE_FIXED_FUNCTION = <TransformType.TRANSFORM_TYPE_FIXED_FUNCTION: 9>

TRANSFORM_TYPE_GRADING_HUE_CURVE = <TransformType.TRANSFORM_TYPE_GRADING_HUE_CURVE: 10>

TRANSFORM_TYPE_GRADING_PRIMARY = <TransformType.TRANSFORM_TYPE_GRADING_PRIMARY: 11>

TRANSFORM_TYPE_GRADING_RGB_CURVE = <TransformType.TRANSFORM_TYPE_GRADING_RGB_CURVE: 12>

TRANSFORM_TYPE_GRADING_TONE = <TransformType.TRANSFORM_TYPE_GRADING_TONE: 13>

TRANSFORM_TYPE_GROUP = <TransformType.TRANSFORM_TYPE_GROUP: 14>

TRANSFORM_TYPE_LOG = <TransformType.TRANSFORM_TYPE_LOG: 17>

TRANSFORM_TYPE_LOG_AFFINE = <TransformType.TRANSFORM_TYPE_LOG_AFFINE: 15>

TRANSFORM_TYPE_LOG_CAMERA = <TransformType.TRANSFORM_TYPE_LOG_CAMERA: 16>

TRANSFORM_TYPE_LOOK = <TransformType.TRANSFORM_TYPE_LOOK: 18>

TRANSFORM_TYPE_LUT1D = <TransformType.TRANSFORM_TYPE_LUT1D: 19>

TRANSFORM_TYPE_LUT3D = <TransformType.TRANSFORM_TYPE_LUT3D: 20>

TRANSFORM_TYPE_MATRIX = <TransformType.TRANSFORM_TYPE_MATRIX: 21>

TRANSFORM_TYPE_RANGE = <TransformType.TRANSFORM_TYPE_RANGE: 22>

property value

C++

enum OpenColorIO_v2_5::TransformType

Values:

enumerator TRANSFORM_TYPE_ALLOCATION

enumerator TRANSFORM_TYPE_BUILTIN

enumerator TRANSFORM_TYPE_CDL

enumerator TRANSFORM_TYPE_COLORSPACE

enumerator TRANSFORM_TYPE_DISPLAY_VIEW

enumerator TRANSFORM_TYPE_EXPONENT

enumerator **TRANSFORM_TYPE_EXPONENT_WITH_LINEAR**

enumerator **TRANSFORM_TYPE_EXPOSURE_CONTRAST**

enumerator **TRANSFORM_TYPE_FILE**

enumerator **TRANSFORM_TYPE_FIXED_FUNCTION**

enumerator **TRANSFORM_TYPE_GRADING_HUE_CURVE**

enumerator **TRANSFORM_TYPE_GRADING_PRIMARY**

enumerator **TRANSFORM_TYPE_GRADING_RGB_CURVE**

enumerator **TRANSFORM_TYPE_GRADING_TONE**

enumerator **TRANSFORM_TYPE_GROUP**

enumerator **TRANSFORM_TYPE_LOG_AFFINE**

enumerator **TRANSFORM_TYPE_LOG_CAMERA**

enumerator **TRANSFORM_TYPE_LOG**

enumerator **TRANSFORM_TYPE_LOOK**

enumerator **TRANSFORM_TYPE_LUT1D**

enumerator **TRANSFORM_TYPE_LUT3D**

enumerator **TRANSFORM_TYPE_MATRIX**

enumerator **TRANSFORM_TYPE_RANGE**

Interpolation

Python

class PyOpenColorIO.**Interpolation**

Specify the interpolation type to use If the specified interpolation type is not supported in the requested context (for example, using tetrahedral interpolationon 1D LUTs) an exception will be thrown.

INTERP_DEFAULT will choose the default interpolation type for the requested context:

1D LUT INTERP_DEFAULT: LINEAR 3D LUT INTERP_DEFAULT: LINEAR

INTERP_BEST will choose the best interpolation type for the requested context:

1D LUT INTERP_BEST: LINEAR 3D LUT INTERP_BEST: TETRAHEDRAL

Note: INTERP_BEST and INTERP_DEFAULT are subject to change in minor releases, so if you care about locking off on a specific interpolation type, we'd recommend directly specifying it.

Members:

INTERP_UNKNOWN :

INTERP_NEAREST : nearest neighbor

INTERP_LINEAR : linear interpolation (trilinear for Lut3D)

INTERP_TETRAHEDRAL : tetrahedral interpolation (Lut3D only)

INTERP_CUBIC : cubic interpolation (not supported)

INTERP_DEFAULT : the default interpolation type

INTERP_BEST : the 'best' suitable interpolation type

INTERP_BEST = <Interpolation.INTERP_BEST: 255>

INTERP_CUBIC = <Interpolation.INTERP_CUBIC: 4>

INTERP_DEFAULT = <Interpolation.INTERP_DEFAULT: 254>

INTERP_LINEAR = <Interpolation.INTERP_LINEAR: 2>

INTERP_NEAREST = <Interpolation.INTERP_NEAREST: 1>

INTERP_TETRAHEDRAL = <Interpolation.INTERP_TETRAHEDRAL: 3>

INTERP_UNKNOWN = <Interpolation.INTERP_UNKNOWN: 0>

property value

C++

enum `OpenColorIO_v2_5::`**Interpolation**

Specify the interpolation type to use If the specified interpolation type is not supported in the requested context (for example, using tetrahedral interpolationon 1D LUTs) an exception will be thrown.

INTERP_DEFAULT will choose the default interpolation type for the requested context:

1D LUT INTERP_DEFAULT: LINEAR 3D LUT INTERP_DEFAULT: LINEAR

INTERP_BEST will choose the best interpolation type for the requested context:

1D LUT INTERP_BEST: LINEAR 3D LUT INTERP_BEST: TETRAHEDRAL

Note: INTERP_BEST and INTERP_DEFAULT are subject to change in minor releases, so if you care about locking off on a specific interpolation type, we'd recommend directly specifying it.

Values:

enumerator **INTERP_UNKNOWN**

enumerator **INTERP_NEAREST**

nearest neighbor

enumerator **INTERP_LINEAR**

linear interpolation (trilinear for Lut3D)

enumerator **INTERP_TETRAHEDRAL**

tetrahedral interpolation (Lut3D only)

enumerator **INTERP_CUBIC**

cubic interpolation (not supported)

enumerator **INTERP_DEFAULT**

the default interpolation type

enumerator **INTERP_BEST**

the 'best' suitable interpolation type

Conversions: *Interpolation*

BitDepth

Python

class PyOpenColorIO.**BitDepth**

Used in a configuration file to indicate the bit-depth of a color space, and by the Processor to specify the input and output bit-depths of images to process. Note that Processor only supports: UINT8, UINT10, UINT12, UINT16, F16 and F32.

Members:

BIT_DEPTH_UNKNOWN :

BIT_DEPTH_UINT8 :

BIT_DEPTH_UINT10 :

BIT_DEPTH_UINT12 :

BIT_DEPTH_UINT14 :

BIT_DEPTH_UINT16 :

BIT_DEPTH_UINT32 : Here for historical reason but that's not supported.

BIT_DEPTH_F16 :

BIT_DEPTH_F32 :

BIT_DEPTH_F16 = <BitDepth.BIT_DEPTH_F16: 7>

BIT_DEPTH_F32 = <BitDepth.BIT_DEPTH_F32: 8>

BIT_DEPTH_UINT10 = <BitDepth.BIT_DEPTH_UINT10: 2>

BIT_DEPTH_UINT12 = <BitDepth.BIT_DEPTH_UINT12: 3>

BIT_DEPTH_UINT14 = <BitDepth.BIT_DEPTH_UINT14: 4>

BIT_DEPTH_UINT16 = <BitDepth.BIT_DEPTH_UINT16: 5>

BIT_DEPTH_UINT32 = <BitDepth.BIT_DEPTH_UINT32: 6>

BIT_DEPTH_UINT8 = <BitDepth.BIT_DEPTH_UINT8: 1>

BIT_DEPTH_UNKNOWN = <BitDepth.BIT_DEPTH_UNKNOWN: 0>

property value

C++

enum OpenColorIO_v2_5::**BitDepth**

Used in a configuration file to indicate the bit-depth of a color space, and by the Processor to specify the input and output bit-depths of images to process. Note that Processor only supports: UINT8, UINT10, UINT12, UINT16, F16 and F32.

Values:

enumerator **BIT_DEPTH_UNKNOWN**

enumerator **BIT_DEPTH_UINT8**

enumerator **BIT_DEPTH_UINT10**

enumerator **BIT_DEPTH_UINT12**

enumerator **BIT_DEPTH_UINT14**

enumerator **BIT_DEPTH_UINT16**

enumerator **BIT_DEPTH_UINT32**

Here for historical reason but that's not supported.

enumerator **BIT_DEPTH_F16**

enumerator **BIT_DEPTH_F32**

Conversions: *BitDepth*

Lut1DHueAdjust

Python

class PyOpenColorIO.**Lut1DHueAdjust**

Used by :cpp:class`:ref:*Lut1DTransform* ` to control optional hue restoration algorithm.

Members:

HUE_NONE : No adjustment.

HUE_DW3 : Algorithm used in ACES Output Transforms through v0.7.

HUE_WYPN : Weighted Yellow Power Norm – NOT IMPLEMENTED YET.

HUE_DW3 = <Lut1DHueAdjust.HUE_DW3: 1>

HUE_NONE = <Lut1DHueAdjust.HUE_NONE: 0>

HUE_WYPN = <Lut1DHueAdjust.HUE_WYPN: 2>

property value

C++

enum OpenColorIO_v2_5::**Lut1DHueAdjust**

Used by :cpp:classLut1DTransform to control optional hue restoration algorithm.

Values:

enumerator **HUE_NONE**

No adjustment.

enumerator **HUE_DW3**

Algorithm used in ACES Output Transforms through v0.7.

enumerator **HUE_WYPN**

Weighted Yellow Power Norm — NOT IMPLEMENTED YET.

ChannelOrdering

Python

class PyOpenColorIO.**ChannelOrdering**

Used by PackedImageDesc to indicate the channel ordering of the image to process.

Members:

CHANNEL_ORDERING_RGBA :

CHANNEL_ORDERING_BGRA :

CHANNEL_ORDERING_ABGR :

CHANNEL_ORDERING_RGB :

CHANNEL_ORDERING_BGR :

CHANNEL_ORDERING_ABGR = <ChannelOrdering.CHANNEL_ORDERING_ABGR: 2>

CHANNEL_ORDERING_BGR = <ChannelOrdering.CHANNEL_ORDERING_BGR: 4>

CHANNEL_ORDERING_BGRA = <ChannelOrdering.CHANNEL_ORDERING_BGRA: 1>

CHANNEL_ORDERING_RGB = <ChannelOrdering.CHANNEL_ORDERING_RGB: 3>

CHANNEL_ORDERING_RGBA = <ChannelOrdering.CHANNEL_ORDERING_RGBA: 0>

property value

C++

enum OpenColorIO_v2_5::**ChannelOrdering**

Used by PackedImageDesc to indicate the channel ordering of the image to process.

Values:

enumerator **CHANNEL_ORDERING_RGBA**

enumerator **CHANNEL_ORDERING_BGRA**

enumerator **CHANNEL_ORDERING_ABGR**

enumerator **CHANNEL_ORDERING_RGB**

enumerator **CHANNEL_ORDERING_BGR**

Allocation

Python

class PyOpenColorIO.**Allocation**

Members:

ALLOCATION_UNKNOWN :

ALLOCATION_UNIFORM :

ALLOCATION_LG2 :

ALLOCATION_LG2 = <Allocation.ALLOCATION_LG2: 2>

ALLOCATION_UNIFORM = <Allocation.ALLOCATION_UNIFORM: 1>

ALLOCATION_UNKNOWN = <Allocation.ALLOCATION_UNKNOWN: 0>

property value

C++

enum OpenColorIO_v2_5::**Allocation**

Values:

enumerator **ALLOCATION_UNKNOWN**

enumerator **ALLOCATION_UNIFORM**

enumerator **ALLOCATION_LG2**

Conversions: *Allocation*

GpuLanguage

Python

class PyOpenColorIO.**GpuLanguage**

Used when there is a choice of hardware shader language.

Members:

GPU_LANGUAGE_CG : Nvidia Cg shader.

GPU_LANGUAGE_GLSL_1_2 : OpenGL Shading Language.

GPU_LANGUAGE_GLSL_1_3 : OpenGL Shading Language.

GPU_LANGUAGE_GLSL_4_0 : OpenGL Shading Language.

GPU_LANGUAGE_GLSL_VK_4_6 : OpenGL Shading Language for Vulkan.

GPU_LANGUAGE_GLSL_ES_1_0 : OpenGL ES Shading Language.

GPU_LANGUAGE_GLSL_ES_3_0 : OpenGL ES Shading Language.

GPU_LANGUAGE_HLSL_DX11 :

GPU_LANGUAGE_MSL_2_0 : Metal Shading Language.

LANGUAGE_OSL_1 : Open Shading Language.

GPU_LANGUAGE_CG = <GpuLanguage.GPU_LANGUAGE_CG: 0>

GPU_LANGUAGE_GLSL_1_2 = <GpuLanguage.GPU_LANGUAGE_GLSL_1_2: 1>

GPU_LANGUAGE_GLSL_1_3 = <GpuLanguage.GPU_LANGUAGE_GLSL_1_3: 2>

GPU_LANGUAGE_GLSL_4_0 = <GpuLanguage.GPU_LANGUAGE_GLSL_4_0: 3>

GPU_LANGUAGE_GLSL_ES_1_0 = <GpuLanguage.GPU_LANGUAGE_GLSL_ES_1_0: 7>

GPU_LANGUAGE_GLSL_ES_3_0 = <GpuLanguage.GPU_LANGUAGE_GLSL_ES_3_0: 8>

GPU_LANGUAGE_GLSL_VK_4_6 = <GpuLanguage.GPU_LANGUAGE_GLSL_VK_4_6: 4>

GPU_LANGUAGE_HLSL_DX11 = <GpuLanguage.GPU_LANGUAGE_HLSL_DX11: 5>

GPU_LANGUAGE_MSL_2_0 = <GpuLanguage.GPU_LANGUAGE_MSL_2_0: 9>

LANGUAGE_OSL_1 = <GpuLanguage.LANGUAGE_OSL_1: 6>

property value

C++

enum OpenColorIO_v2_5::**GpuLanguage**

Used when there is a choice of hardware shader language.

Values:

enumerator **GPU_LANGUAGE_CG**

Nvidia Cg shader.

enumerator **GPU_LANGUAGE_GLSL_1_2**

OpenGL Shading Language.

enumerator **GPU_LANGUAGE_GLSL_1_3**

OpenGL Shading Language.

enumerator **GPU_LANGUAGE_GLSL_4_0**

OpenGL Shading Language.

enumerator **GPU_LANGUAGE_GLSL_VK_4_6**

OpenGL Shading Language for Vulkan.

enumerator **GPU_LANGUAGE_HLSL_SM_5_0**

DirectX High Level Shading Language.

enumerator **LANGUAGE_OSL_1**

Open Shading Language.

enumerator **GPU_LANGUAGE_GLSL_ES_1_0**

OpenGL ES Shading Language.

enumerator **GPU_LANGUAGE_GLSL_ES_3_0**

OpenGL ES Shading Language.

enumerator **GPU_LANGUAGE_MSL_2_0**

Metal Shading Language.

enumerator **GPU_LANGUAGE_HLSL_DX11**

Conversions: *GpuLanguage*

EnvironmentMode

Python

class PyOpenColorIO.**EnvironmentMode**

Controls which environment variables are loaded into a Context object.

Members:

ENV_ENVIRONMENT_UNKNOWN :

ENV_ENVIRONMENT_LOAD_PREDEFINED : Only load vars in the config's environment section.

ENV_ENVIRONMENT_LOAD_ALL : Load all env. vars (note: may reduce performance)

ENV_ENVIRONMENT_LOAD_ALL = <EnvironmentMode.ENV_ENVIRONMENT_LOAD_ALL: 2>

ENV_ENVIRONMENT_LOAD_PREDEFINED = <EnvironmentMode.ENV_ENVIRONMENT_LOAD_PREDEFINED: 1>

ENV_ENVIRONMENT_UNKNOWN = <EnvironmentMode.ENV_ENVIRONMENT_UNKNOWN: 0>

property value

C++

enum OpenColorIO_v2_5::**EnvironmentMode**

Controls which environment variables are loaded into a Context object.

Values:

enumerator **ENV_ENVIRONMENT_UNKNOWN**

enumerator **ENV_ENVIRONMENT_LOAD_PREDEFINED**

Only load vars in the config's environment section.

enumerator **ENV_ENVIRONMENT_LOAD_ALL**

Load all env. vars (note: may reduce performance)

Conversions: *EnvironmentMode*

RangeStyle

Python

class PyOpenColorIO.**RangeStyle**

A RangeTransform may be set to clamp the values, or not.

Members:

RANGE_NO_CLAMP :

RANGE_CLAMP :

RANGE_CLAMP = <RangeStyle.RANGE_CLAMP: 1>

RANGE_NO_CLAMP = <RangeStyle.RANGE_NO_CLAMP: 0>

property value

C++

enum OpenColorIO_v2_5::**RangeStyle**

A RangeTransform may be set to clamp the values, or not.

Values:

enumerator **RANGE_NO_CLAMP**

enumerator **RANGE_CLAMP**

Conversions: *RangeStyle*

FixedFunctionStyle

Python

class PyOpenColorIO.**FixedFunctionStyle**

Enumeration of the :ref:`FixedFunctionTransform` transform algorithms.

Members:

FIXED_FUNCTION_ACES_RED_MOD_03 : Red modifier (ACES 0.3/0.7)

FIXED_FUNCTION_ACES_RED_MOD_10 : Red modifier (ACES 1.0)

FIXED_FUNCTION_ACES_GLOW_03 : Glow function (ACES 0.3/0.7)

FIXED_FUNCTION_ACES_GLOW_10 : Glow function (ACES 1.0)

FIXED_FUNCTION_ACES_DARK_TO_DIM_10 : Dark to dim surround correction (ACES 1.0)

FIXED_FUNCTION_REC2100_SURROUND : Rec.2100 surround correction (takes one double for the gamma param)

FIXED_FUNCTION_RGB_TO_HSV : Classic RGB to HSV function.

FIXED_FUNCTION_XYZ_TO_xyY : CIE XYZ to 1931 xy chromaticity coordinates.

FIXED_FUNCTION_XYZ_TO_uvY : CIE XYZ to 1976 u'v' chromaticity coordinates.

FIXED_FUNCTION_XYZ_TO_LUV : CIE XYZ to 1976 CIELUV colour space (D65 white)

FIXED_FUNCTION_ACES_GAMUTMAP_02 : ACES 0.2 Gamut clamping algorithm – NOT IMPLEMENTED YET.

FIXED_FUNCTION_ACES_GAMUTMAP_07 : ACES 0.7 Gamut clamping algorithm – NOT IMPLEMENTED YET.

FIXED_FUNCTION_ACES_GAMUT_COMP_13 : ACES 1.3 Parametric Gamut Compression (expects ACEScg values)

FIXED_FUNCTION_LIN_TO_PQ : SMPTE ST-2084 OETF, scaled with 100 nits at 1.0 (neg vals mirrored)

FIXED_FUNCTION_LIN_TO_GAMMA_LOG : Parametrized gamma and log segments with mirroring.

FIXED_FUNCTION_LIN_TO_DOUBLE_LOG : Two parameterized LogAffineTransforms with a middle linear segment.

FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20 : ACES 2.0 Display Rendering – EXPERIMENTAL.

FIXED_FUNCTION_ACES_RGB_TO_JMH_20 : ACES 2.0 RGB to JMh – EXPERIMENTAL.

FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20 : ACES 2.0 Tonescale and chroma compression – EXPERIMENTAL.

FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20 : ACES 2.0 Gamut compression – EXPERIMENTAL.

FIXED_FUNCTION_RGB_TO_HSY_LIN : RGB to HSY (Hue, Saturation, Luminance) for linear spaces.

FIXED_FUNCTION_RGB_TO_HSY_LOG : RGB to HSY (Hue, Saturation, Luma) for log spaces.

FIXED_FUNCTION_RGB_TO_HSY_VID : RGB to HSY (Hue, Saturation, Luma) for video spaces.

```
FIXED_FUNCTION_ACES_DARK_TO_DIM_10 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_DARK_TO_DIM_10:  4>

FIXED_FUNCTION_ACES_GAMUTMAP_02 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_GAMUTMAP_02:  10>

FIXED_FUNCTION_ACES_GAMUTMAP_07 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_GAMUTMAP_07:  11>

FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20:  19>

FIXED_FUNCTION_ACES_GAMUT_COMP_13 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_GAMUT_COMP_13:  12>

FIXED_FUNCTION_ACES_GLOW_03 = <FixedFunctionStyle.FIXED_FUNCTION_ACES_GLOW_03:  2>

FIXED_FUNCTION_ACES_GLOW_10 = <FixedFunctionStyle.FIXED_FUNCTION_ACES_GLOW_10:  3>

FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20:  16>

FIXED_FUNCTION_ACES_RED_MOD_03 = <FixedFunctionStyle.FIXED_FUNCTION_ACES_RED_MOD_03:
0>

FIXED_FUNCTION_ACES_RED_MOD_10 = <FixedFunctionStyle.FIXED_FUNCTION_ACES_RED_MOD_10:
1>

FIXED_FUNCTION_ACES_RGB_TO_JMH_20 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_RGB_TO_JMH_20:  17>

FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20 =
<FixedFunctionStyle.FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20:  18>

FIXED_FUNCTION_LIN_TO_DOUBLE_LOG =
<FixedFunctionStyle.FIXED_FUNCTION_LIN_TO_DOUBLE_LOG: 15>

FIXED_FUNCTION_LIN_TO_GAMMA_LOG =
<FixedFunctionStyle.FIXED_FUNCTION_LIN_TO_GAMMA_LOG: 14>

FIXED_FUNCTION_LIN_TO_PQ = <FixedFunctionStyle.FIXED_FUNCTION_LIN_TO_PQ: 13>

FIXED_FUNCTION_REC2100_SURROUND =
<FixedFunctionStyle.FIXED_FUNCTION_REC2100_SURROUND: 5>

FIXED_FUNCTION_RGB_TO_HSV = <FixedFunctionStyle.FIXED_FUNCTION_RGB_TO_HSV: 6>

FIXED_FUNCTION_RGB_TO_HSY_LIN = <FixedFunctionStyle.FIXED_FUNCTION_RGB_TO_HSY_LIN:
20>

FIXED_FUNCTION_RGB_TO_HSY_LOG = <FixedFunctionStyle.FIXED_FUNCTION_RGB_TO_HSY_LOG:
21>

FIXED_FUNCTION_RGB_TO_HSY_VID = <FixedFunctionStyle.FIXED_FUNCTION_RGB_TO_HSY_VID:
22>

FIXED_FUNCTION_XYZ_TO_LUV = <FixedFunctionStyle.FIXED_FUNCTION_XYZ_TO_LUV: 9>
```

FIXED_FUNCTION_XYZ_TO_uvY = <FixedFunctionStyle.FIXED_FUNCTION_XYZ_TO_uvY: 8>

FIXED_FUNCTION_XYZ_TO_xyY = <FixedFunctionStyle.FIXED_FUNCTION_XYZ_TO_xyY: 7>

property value

C++

enum `OpenColorIO_v2_5::`**FixedFunctionStyle**

Enumeration of the :cpp:class:`FixedFunctionTransform` transform algorithms.

Values:

enumerator **FIXED_FUNCTION_ACES_RED_MOD_03**

Red modifier (ACES 0.3/0.7)

enumerator **FIXED_FUNCTION_ACES_RED_MOD_10**

Red modifier (ACES 1.0)

enumerator **FIXED_FUNCTION_ACES_GLOW_03**

Glow function (ACES 0.3/0.7)

enumerator **FIXED_FUNCTION_ACES_GLOW_10**

Glow function (ACES 1.0)

enumerator **FIXED_FUNCTION_ACES_DARK_TO_DIM_10**

Dark to dim surround correction (ACES 1.0)

enumerator **FIXED_FUNCTION_REC2100_SURROUND**

Rec.2100 surround correction (takes one double for the gamma param)

enumerator **FIXED_FUNCTION_RGB_TO_HSV**

Classic RGB to HSV function.

enumerator **FIXED_FUNCTION_XYZ_TO_xyY**

CIE XYZ to 1931 xy chromaticity coordinates.

enumerator **FIXED_FUNCTION_XYZ_TO_uvY**

CIE XYZ to 1976 u'v' chromaticity coordinates.

enumerator **FIXED_FUNCTION_XYZ_TO_LUV**

CIE XYZ to 1976 CIELUV colour space (D65 white)

enumerator **FIXED_FUNCTION_ACES_GAMUTMAP_02**

ACES 0.2 Gamut clamping algorithm — NOT IMPLEMENTED YET.

enumerator **FIXED_FUNCTION_ACES_GAMUTMAP_07**

ACES 0.7 Gamut clamping algorithm — NOT IMPLEMENTED YET.

enumerator **FIXED_FUNCTION_ACES_GAMUT_COMP_13**

ACES 1.3 Parametric Gamut Compression (expects ACEScg values)

enumerator **FIXED_FUNCTION_LIN_TO_PQ**

SMPTE ST-2084 OETF, scaled with 100 nits at 1.0 (neg vals mirrored)

enumerator **FIXED_FUNCTION_LIN_TO_GAMMA_LOG**

Parametrized gamma and log segments with mirroring.

enumerator **FIXED_FUNCTION_LIN_TO_DOUBLE_LOG**

Two parameterized LogAffineTransforms with a middle linear segment.

enumerator **FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20**

ACES 2.0 Display Rendering — EXPERIMENTAL.

enumerator **FIXED_FUNCTION_ACES_RGB_TO_JMH_20**

ACES 2.0 RGB to JMh — EXPERIMENTAL.

enumerator **FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20**

ACES 2.0 Tonescale and chroma compression — EXPERIMENTAL.

enumerator **FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20**

ACES 2.0 Gamut compression — EXPERIMENTAL.

enumerator **FIXED_FUNCTION_RGB_TO_HSY_LIN**

RGB to HSY (Hue, Saturation, Luminance) for linear spaces.

enumerator **FIXED_FUNCTION_RGB_TO_HSY_LOG**

RGB to HSY (Hue, Saturation, Luma) for log spaces.

enumerator **FIXED_FUNCTION_RGB_TO_HSY_VID**

RGB to HSY (Hue, Saturation, Luma) for video spaces.

Conversions: *FixedFunctionStyle*

ExposureContrastStyle

Python

class PyOpenColorIO.**ExposureContrastStyle**

Enumeration of the :ref:`ExposureContrastTransform` transform algorithms.

Members:

EXPOSURE_CONTRAST_LINEAR : E/C to be applied to a linear space image.

EXPOSURE_CONTRAST_VIDEO : E/C to be applied to a video space image.

EXPOSURE_CONTRAST_LOGARITHMIC : E/C to be applied to a log space image.

EXPOSURE_CONTRAST_LINEAR = <ExposureContrastStyle.EXPOSURE_CONTRAST_LINEAR: 0>

EXPOSURE_CONTRAST_LOGARITHMIC = <ExposureContrastStyle.EXPOSURE_CONTRAST_LOGARITHMIC: 2>

EXPOSURE_CONTRAST_VIDEO = <ExposureContrastStyle.EXPOSURE_CONTRAST_VIDEO: 1>

property value

C++

enum OpenColorIO_v2_5::**ExposureContrastStyle**

Enumeration of the :cpp:class:ExposureContrastTransform transform algorithms.

Values:

enumerator **EXPOSURE_CONTRAST_LINEAR**

E/C to be applied to a linear space image.

enumerator **EXPOSURE_CONTRAST_VIDEO**

E/C to be applied to a video space image.

enumerator **EXPOSURE_CONTRAST_LOGARITHMIC**

E/C to be applied to a log space image.

Conversions: *ExposureContrastStyle*

CDLStyle

Python

class PyOpenColorIO.**CDLStyle**

Members:

CDL_ASC : ASC CDL specification v1.2.

CDL_NO_CLAMP : CDL that does not clamp.

CDL_TRANSFORM_DEFAULT :

CDL_ASC = <CDLStyle.CDL_ASC: 0>

CDL_NO_CLAMP = <CDLStyle.CDL_NO_CLAMP: 1>

CDL_TRANSFORM_DEFAULT = <CDLStyle.CDL_NO_CLAMP: 1>

property value

C++

enum OpenColorIO_v2_5::**CDLStyle**

Enumeration of the :cpp:class:CDLTransform transform algorithms.

Note: The default for reading .cc/.ccc/.cdl files, config file YAML, and CDLTransform is no-clamp, since that is what is primarily desired in VFX. However, the CLF format default is ASC.

Values:

enumerator **CDL_ASC**

ASC CDL specification v1.2.

enumerator **CDL_NO_CLAMP**

CDL that does not clamp.

enumerator **CDL_TRANSFORM_DEFAULT**

Conversions: *CDLStyle*

NegativeStyle

Python

class PyOpenColorIO.**NegativeStyle**

Negative values handling style for ExponentTransform and ExponentWithLinearTransform transform algorithms.

Members:

NEGATIVE_CLAMP : Clamp negative values.

NEGATIVE_MIRROR : Positive curve is rotated 180 degrees around the origin to handle negatives.

NEGATIVE_PASS_THRU : Negative values are passed through unchanged.

NEGATIVE_LINEAR : Linearly extrapolate the curve for negative values.

NEGATIVE_CLAMP = <NegativeStyle.NEGATIVE_CLAMP: 0>

NEGATIVE_LINEAR = <NegativeStyle.NEGATIVE_LINEAR: 3>

NEGATIVE_MIRROR = <NegativeStyle.NEGATIVE_MIRROR: 1>

NEGATIVE_PASS_THRU = <NegativeStyle.NEGATIVE_PASS_THRU: 2>

property value

C++

enum OpenColorIO_v2_5::**NegativeStyle**

Negative values handling style for ExponentTransform and ExponentWithLinearTransform transform algorithms.

Values:

enumerator **NEGATIVE_CLAMP**

Clamp negative values.

enumerator **NEGATIVE_MIRROR**

Positive curve is rotated 180 degrees around the origin to handle negatives.

enumerator **NEGATIVE_PASS_THRU**

Negative values are passed through unchanged.

enumerator **NEGATIVE_LINEAR**

Linearly extrapolate the curve for negative values.

Conversions: *NegativeStyle*

GradingStyle

Python

class PyOpenColorIO.**GradingStyle**

Styles for grading transforms.

Members:

GRADING_LOG : Algorithms for Logarithmic color spaces.

GRADING_LIN : Algorithms for Scene Linear color spaces.

GRADING_VIDEO : Algorithms for Video color spaces.

GRADING_LIN = <GradingStyle.GRADING_LIN: 1>

GRADING_LOG = <GradingStyle.GRADING_LOG: 0>

GRADING_VIDEO = <GradingStyle.GRADING_VIDEO: 2>

property value

C++

enum OpenColorIO_v2_5::**GradingStyle**

Styles for grading transforms.

Values:

enumerator **GRADING_LOG**

Algorithms for Logarithmic color spaces.

enumerator **GRADING_LIN**

Algorithms for Scene Linear color spaces.

enumerator **GRADING_VIDEO**

Algorithms for Video color spaces.

Conversions: *GradingStyle*

DynamicPropertyType

Python

class PyOpenColorIO.**DynamicPropertyType**

Types for dynamic properties.

Members:

DYNAMIC_PROPERTY_EXPOSURE : Image exposure value (double floating point value)

DYNAMIC_PROPERTY_CONTRAST : Image contrast value (double floating point value)

DYNAMIC_PROPERTY_GAMMA : Image gamma value (double floating point value)

DYNAMIC_PROPERTY_GRADING_PRIMARY : Used by GradingPrimaryTransform.

DYNAMIC_PROPERTY_GRADING_RGBCURVE : Used by GradingRGBCurveTransform.

DYNAMIC_PROPERTY_GRADING_HUECURVE : Used by GradingHueCurveTransform.

DYNAMIC_PROPERTY_GRADING_TONE : Used by GradingToneTransform.

DYNAMIC_PROPERTY_CONTRAST = <DynamicPropertyType.DYNAMIC_PROPERTY_CONTRAST: 1>

DYNAMIC_PROPERTY_EXPOSURE = <DynamicPropertyType.DYNAMIC_PROPERTY_EXPOSURE: 0>

DYNAMIC_PROPERTY_GAMMA = <DynamicPropertyType.DYNAMIC_PROPERTY_GAMMA: 2>

DYNAMIC_PROPERTY_GRADING_HUECURVE =
<DynamicPropertyType.DYNAMIC_PROPERTY_GRADING_HUECURVE: 6>

DYNAMIC_PROPERTY_GRADING_PRIMARY =
<DynamicPropertyType.DYNAMIC_PROPERTY_GRADING_PRIMARY: 3>

DYNAMIC_PROPERTY_GRADING_RGBCURVE =
<DynamicPropertyType.DYNAMIC_PROPERTY_GRADING_RGBCURVE: 4>

DYNAMIC_PROPERTY_GRADING_TONE = <DynamicPropertyType.DYNAMIC_PROPERTY_GRADING_TONE: 5>

property value

C++

enum OpenColorIO_v2_5::**DynamicPropertyType**

Types for dynamic properties.

Values:

enumerator **DYNAMIC_PROPERTY_EXPOSURE**

Image exposure value (double floating point value)

enumerator **DYNAMIC_PROPERTY_CONTRAST**

Image contrast value (double floating point value)

enumerator **DYNAMIC_PROPERTY_GAMMA**

Image gamma value (double floating point value)

enumerator **DYNAMIC_PROPERTY_GRADING_PRIMARY**

Used by GradingPrimaryTransform.

enumerator **DYNAMIC_PROPERTY_GRADING_RGBCURVE**

Used by GradingRGBCurveTransform.

enumerator **DYNAMIC_PROPERTY_GRADING_TONE**

Used by GradingToneTransform.

enumerator **DYNAMIC_PROPERTY_GRADING_HUECURVE**

Used by GradingHueCurveTransform.

RGBCurveType

Python

class PyOpenColorIO.**RGBCurveType**

Types for GradingRGBCurve.

Members:

RGB_RED :

RGB_GREEN :

RGB_BLUE :

RGB_MASTER :

RGB_NUM_CURVES :

RGB_BLUE = <RGBCurveType.RGB_BLUE: 2>

RGB_GREEN = <RGBCurveType.RGB_GREEN: 1>

RGB_MASTER = <RGBCurveType.RGB_MASTER: 3>

RGB_NUM_CURVES = <RGBCurveType.RGB_NUM_CURVES: 4>

RGB_RED = <RGBCurveType.RGB_RED: 0>

property value

C++

enum OpenColorIO_v2_5::**RGBCurveType**

Types for GradingRGBCurve.

Values:

enumerator **RGB_RED**

enumerator **RGB_GREEN**

enumerator **RGB_BLUE**

enumerator **RGB_MASTER**

enumerator **RGB_NUM_CURVES**

UniformDataType

Python

class PyOpenColorIO.**UniformDataType**

Types for uniform data.

Members:

UNIFORM_DOUBLE :

UNIFORM_BOOL :

UNIFORM_FLOAT3 : Array of 3 floats.

UNIFORM_VECTOR_FLOAT : Vector of floats (size is set by uniform).

UNIFORM_VECTOR_INT : Vector of int pairs (size is set by uniform).

UNIFORM_UNKNOWN :

UNIFORM_BOOL = <UniformDataType.UNIFORM_BOOL: 1>

UNIFORM_DOUBLE = <UniformDataType.UNIFORM_DOUBLE: 0>

UNIFORM_FLOAT3 = <UniformDataType.UNIFORM_FLOAT3: 2>

UNIFORM_UNKNOWN = <UniformDataType.UNIFORM_UNKNOWN: 5>

UNIFORM_VECTOR_FLOAT = <UniformDataType.UNIFORM_VECTOR_FLOAT: 3>

UNIFORM_VECTOR_INT = <UniformDataType.UNIFORM_VECTOR_INT: 4>

property value

C++

enum OpenColorIO_v2_5::**UniformDataType**

Types for uniform data.

Values:

enumerator **UNIFORM_DOUBLE**

enumerator **UNIFORM_BOOL**

enumerator **UNIFORM_FLOAT3**

Array of 3 floats.

enumerator **UNIFORM_VECTOR_FLOAT**

Vector of floats (size is set by uniform).

enumerator **UNIFORM_VECTOR_INT**

Vector of int pairs (size is set by uniform).

enumerator **UNIFORM_UNKNOWN**

OptimizationFlags

Python

class PyOpenColorIO.**OptimizationFlags**

Provides control over how the ops in a Processor are combined in order to improve performance.

Members:

OPTIMIZATION_NONE : Do not optimize.

OPTIMIZATION_IDENTITY : Replace identity ops (other than gamma).

OPTIMIZATION_IDENTITY_GAMMA : Replace identity gamma ops.

OPTIMIZATION_PAIR_IDENTITY_CDL : Replace a pair of ops where one is the inverse of the other.

OPTIMIZATION_PAIR_IDENTITY_EXPOSURE_CONTRAST :

OPTIMIZATION_PAIR_IDENTITY_FIXED_FUNCTION :

OPTIMIZATION_PAIR_IDENTITY_GAMMA :

OPTIMIZATION_PAIR_IDENTITY_LUT1D :

OPTIMIZATION_PAIR_IDENTITY_LUT3D :

OPTIMIZATION_PAIR_IDENTITY_LOG :

OPTIMIZATION_PAIR_IDENTITY_GRADING :

OPTIMIZATION_COMP_EXPONENT : Compose a pair of ops into a single op.

OPTIMIZATION_COMP_GAMMA :

OPTIMIZATION_COMP_MATRIX :

OPTIMIZATION_COMP_LUT1D :

OPTIMIZATION_COMP_LUT3D :

OPTIMIZATION_COMP_RANGE :

OPTIMIZATION_COMP_SEPARABLE_PREFIX : For integer and half bit-depths only, replace separable ops (i.e. no channel crosstalk ops) by a single 1D LUT of input bit-depth domain.

OPTIMIZATION_LUT_INV_FAST : Implement inverse Lut1D and Lut3D evaluations using a a forward LUT (faster but less accurate). Note that GPU evals always do FAST.

OPTIMIZATION_FAST_LOG_EXP_POW :

OPTIMIZATION_SIMPLIFY_OPS :

OPTIMIZATION_NO_DYNAMIC_PROPERTIES : Turn off dynamic control of any ops that offer adjustment of parameter values after finalization (e.g. ExposureContrast).

OPTIMIZATION_ALL : Apply all possible optimizations.

OPTIMIZATION_LOSSLESS :

OPTIMIZATION_VERY_GOOD :

OPTIMIZATION_GOOD :

OPTIMIZATION_DRAFT : For quite lossy optimizations.

OPTIMIZATION_DEFAULT :

OPTIMIZATION_ALL = <OptimizationFlags.OPTIMIZATION_ALL: 4294967295>

OPTIMIZATION_COMP_EXPONENT = <OptimizationFlags.OPTIMIZATION_COMP_EXPONENT: 262144>

OPTIMIZATION_COMP_GAMMA = <OptimizationFlags.OPTIMIZATION_COMP_GAMMA: 524288>

OPTIMIZATION_COMP_LUT1D = <OptimizationFlags.OPTIMIZATION_COMP_LUT1D: 2097152>

OPTIMIZATION_COMP_LUT3D = <OptimizationFlags.OPTIMIZATION_COMP_LUT3D: 4194304>

OPTIMIZATION_COMP_MATRIX = <OptimizationFlags.OPTIMIZATION_COMP_MATRIX: 1048576>

OPTIMIZATION_COMP_RANGE = <OptimizationFlags.OPTIMIZATION_COMP_RANGE: 8388608>

OPTIMIZATION_COMP_SEPARABLE_PREFIX =
<OptimizationFlags.OPTIMIZATION_COMP_SEPARABLE_PREFIX: 16777216>

OPTIMIZATION_DEFAULT = <OptimizationFlags.OPTIMIZATION_VERY_GOOD: 263995331>

OPTIMIZATION_DRAFT = <OptimizationFlags.OPTIMIZATION_ALL: 4294967295>

OPTIMIZATION_FAST_LOG_EXP_POW = <OptimizationFlags.OPTIMIZATION_FAST_LOG_EXP_POW:
67108864>

OPTIMIZATION_GOOD = <OptimizationFlags.OPTIMIZATION_GOOD: 268189635>

OPTIMIZATION_IDENTITY = <OptimizationFlags.OPTIMIZATION_IDENTITY: 1>

OPTIMIZATION_IDENTITY_GAMMA = <OptimizationFlags.OPTIMIZATION_IDENTITY_GAMMA: 2>

OPTIMIZATION_LOSSLESS = <OptimizationFlags.OPTIMIZATION_LOSSLESS: 144457667>

OPTIMIZATION_LUT_INV_FAST = <OptimizationFlags.OPTIMIZATION_LUT_INV_FAST: 33554432>

OPTIMIZATION_NONE = <OptimizationFlags.OPTIMIZATION_NONE: 0>

OPTIMIZATION_NO_DYNAMIC_PROPERTIES =
<OptimizationFlags.OPTIMIZATION_NO_DYNAMIC_PROPERTIES: 268435456>

OPTIMIZATION_PAIR_IDENTITY_CDL = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_CDL:
64>

OPTIMIZATION_PAIR_IDENTITY_EXPOSURE_CONTRAST =
<OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_EXPOSURE_CONTRAST: 128>

OPTIMIZATION_PAIR_IDENTITY_FIXED_FUNCTION = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_FIXED_FUNCTION: 256>

OPTIMIZATION_PAIR_IDENTITY_GAMMA = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_GAMMA: 512>

OPTIMIZATION_PAIR_IDENTITY_GRADING = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_GRADING: 8192>

OPTIMIZATION_PAIR_IDENTITY_LOG = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_LOG: 4096>

OPTIMIZATION_PAIR_IDENTITY_LUT1D = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_LUT1D: 1024>

OPTIMIZATION_PAIR_IDENTITY_LUT3D = <OptimizationFlags.OPTIMIZATION_PAIR_IDENTITY_LUT3D: 2048>

OPTIMIZATION_SIMPLIFY_OPS = <OptimizationFlags.OPTIMIZATION_SIMPLIFY_OPS: 134217728>

OPTIMIZATION_VERY_GOOD = <OptimizationFlags.OPTIMIZATION_VERY_GOOD: 263995331>

property value

C++

enum `OpenColorIO_v2_5::`**OptimizationFlags**

Provides control over how the ops in a Processor are combined in order to improve performance.

Values:

enumerator **OPTIMIZATION_NONE**

Do not optimize.

enumerator **OPTIMIZATION_IDENTITY**

Replace identity ops (other than gamma).

enumerator **OPTIMIZATION_IDENTITY_GAMMA**

Replace identity gamma ops.

enumerator **OPTIMIZATION_PAIR_IDENTITY_CDL**

Replace a pair of ops where one is the inverse of the other.

enumerator **OPTIMIZATION_PAIR_IDENTITY_EXPOSURE_CONTRAST**

enumerator **OPTIMIZATION_PAIR_IDENTITY_FIXED_FUNCTION**

enumerator **OPTIMIZATION_PAIR_IDENTITY_GAMMA**

enumerator **OPTIMIZATION_PAIR_IDENTITY_LUT1D**

enumerator **OPTIMIZATION_PAIR_IDENTITY_LUT3D**

enumerator **OPTIMIZATION_PAIR_IDENTITY_LOG**

enumerator **OPTIMIZATION_PAIR_IDENTITY_GRADING**

enumerator **OPTIMIZATION_COMP_EXPONENT**

Compose a pair of ops into a single op.

enumerator **OPTIMIZATION_COMP_GAMMA**

enumerator **OPTIMIZATION_COMP_MATRIX**

enumerator **OPTIMIZATION_COMP_LUT1D**

enumerator **OPTIMIZATION_COMP_LUT3D**

enumerator **OPTIMIZATION_COMP_RANGE**

enumerator **OPTIMIZATION_COMP_SEPARABLE_PREFIX**

For integer and half bit-depths only, replace separable ops (i.e. no channel crosstalk ops) by a single 1D LUT of input bit-depth domain.

enumerator **OPTIMIZATION_LUT_INV_FAST**

Implement inverse Lut1D and Lut3D evaluations using a a forward LUT (faster but less accurate). Note that GPU evals always do FAST.

enumerator **OPTIMIZATION_FAST_LOG_EXP_POW**

enumerator **OPTIMIZATION_SIMPLIFY_OPS**

enumerator **OPTIMIZATION_NO_DYNAMIC_PROPERTIES**

Turn off dynamic control of any ops that offer adjustment of parameter values after finalization (e.g. ExposureContrast).

enumerator **OPTIMIZATION_ALL**

Apply all possible optimizations.

enumerator **OPTIMIZATION_LOSSLESS**

enumerator **OPTIMIZATION_VERY_GOOD**

enumerator **OPTIMIZATION_GOOD**

enumerator **OPTIMIZATION_DRAFT**

For quite lossy optimizations.

enumerator **OPTIMIZATION_DEFAULT**

ProcessorCacheFlags

Python

class PyOpenColorIO.**ProcessorCacheFlags**

cpp:type:: Enum to control the behavior of the internal caches e.g. the processor cache in

Members:

PROCESSOR_CACHE_OFF :

PROCESSOR_CACHE_ENABLED :

PROCESSOR_CACHE_SHARE_DYN_PROPERTIES :

PROCESSOR_CACHE_DEFAULT :

PROCESSOR_CACHE_DEFAULT = <ProcessorCacheFlags.PROCESSOR_CACHE_DEFAULT: 3>

PROCESSOR_CACHE_ENABLED = <ProcessorCacheFlags.PROCESSOR_CACHE_ENABLED: 1>

PROCESSOR_CACHE_OFF = <ProcessorCacheFlags.PROCESSOR_CACHE_OFF: 0>

PROCESSOR_CACHE_SHARE_DYN_PROPERTIES =
<ProcessorCacheFlags.PROCESSOR_CACHE_SHARE_DYN_PROPERTIES: 2>

property value

C++

enum OpenColorIO_v2_5::**ProcessorCacheFlags**

cpp:type:: Enum to control the behavior of the internal caches e.g. the processor cache in

Values:

enumerator **PROCESSOR_CACHE_OFF**

enumerator **PROCESSOR_CACHE_ENABLED**

enumerator **PROCESSOR_CACHE_SHARE_DYN_PROPERTIES**

enumerator **PROCESSOR_CACHE_DEFAULT**

5.7.15 Exceptions

Python

exception PyOpenColorIO.**Exception**

An exception class to throw for errors detected at runtime.

> **Warning:** All functions in the Config class can potentially throw this exception.

exception PyOpenColorIO.**ExceptionMissingFile**

An exception class for errors detected at runtime.

Thrown when OCIO cannot find a file that is expected to exist. This is provided as a custom type to distinguish cases where one wants to continue looking for missing files, but wants to properly fail for other error conditions.

C++

class **Exception** : public runtime_error

An exception class to throw for errors detected at runtime.

> **Warning:** All functions in the Config class can potentially throw this exception.

Subclassed by ExceptionMissingFile

Public Functions

Exception() = delete

explicit **Exception**(const char*)

Constructor that takes a string as the exception message.

Exception(const *Exception*&)

Constructor that takes an existing exception.

Exception &**operator=**(const *Exception*&) = delete

~Exception()

class **ExceptionMissingFile** : public *Exception*

An exception class for errors detected at runtime.

Thrown when OCIO cannot find a file that is expected to exist. This is provided as a custom type to distinguish cases where one wants to continue looking for missing files, but wants to properly fail for other error conditions.

Public Functions

ExceptionMissingFile() = delete

explicit **ExceptionMissingFile**(const char*)

Constructor that takes a string as the exception message.

ExceptionMissingFile(const *ExceptionMissingFile*&)

Constructor that takes an existing exception.

ExceptionMissingFile &**operator=**(const *ExceptionMissingFile*&) = delete

~ExceptionMissingFile()

5.7.16 FormatMetadata

Python

class PyOpenColorIO.**FormatMetadata**

The FormatMetadata class is intended to be a generic container to hold metadata from various file formats.

This class provides a hierarchical metadata container. A metadata object is similar to an element in XML. The top level element is named "ROOT" and can't be renamed. Several transforms have a FormatMetadata. The root element and all of the sub-elements may contain: - A name string (e.g. "ROOT", "Description"...). Name can't be empty. - A value string (e.g. "updated viewing LUT"). Value can be empty. - A list of attributes (name, value) string pairs (e.g. "version", "1.5"). There are helper functions to get and set "id" and "name" attributes. Attribute names are unique. - And a list of child sub-elements, which are also objects implementing FormatMetadata. There can be several sub-elements with the same name.

name: str **in** *self* → bool

self [*name: str*] → str

Get the value of a attribute ("" if attribute does not exist).

FormatMetadata(**args*, ***kwargs*)

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.AttributeNameIterator*

len(*self*) → int

self [*name: str*] = *value: str* → None

Add an attribute with a given name and value. If an attribute with the same name already exists, its value is replaced. Throw if name is NULL or empty.

addChildElement(*name: str*, *value: str*) → None

Add a child element with a given name and value.

Name has to be non-empty. Value may be empty, particularly if this element will have children. Element is added after all existing children. Use getChildElement(getNumChildrenElements()-1) to access the added element.

clear() → None

Remove all children, all attributes and the value.

getAttributes() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.AttributeIterator*

getChildElements() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.ChildElementIterator*

getElementName() → str

getElementValue() → str

getID() → str

Convenience method to easily get/set the 'id' attribute. This corresponds to the ProcessNode id attribute from a CLF/CTF file or the ColorCorrection id attribute from a CC/CCC/CDL file.

getName() → str

Convenience method to easily get/set the 'name' attribute. This corresponds to the ProcessNode name attribute from a CLF / CTF file or the name key of a transform in the config YAML.

setElementName(*name: str*) → None

Name has to be a non-empty string. Top-level element can't be renamed. 'ROOT' is reserved.

setElementValue(*value: str*) → None

setID(*id: str*) → None

setName(*name: str*) → None

class PyOpenColorIO.FormatMetadata.**AttributeNameIterator**

self[*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.AttributeNameIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.FormatMetadata.**AttributeIterator**

self[*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.AttributeIterator*

len(*self*) → int

next(*self*) → tuple

class PyOpenColorIO.FormatMetadata.**ConstChildElementIterator**

self[*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.ConstChildElementIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

class PyOpenColorIO.FormatMetadata.**ChildElementIterator**

self[*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata.ChildElementIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

C++

class **FormatMetadata**

The FormatMetadata class is intended to be a generic container to hold metadata from various file formats.

This class provides a hierarchical metadata container. A metadata object is similar to an element in XML. The top level element is named "ROOT" and can't be renamed. Several transforms have a FormatMetadata. The root element and all of the sub-elements may contain:

- A name string (e.g. "ROOT", "Description"...). Name can't be empty.
- A value string (e.g. "updated viewing LUT"). Value can be empty.
- A list of attributes (name, value) string pairs (e.g. "version", "1.5"). There are helper functions to get and set "id" and "name" attributes. Attribute names are unique.
- And a list of child sub-elements, which are also objects implementing FormatMetadata. There can be several sub-elements with the same name.

Public Functions

virtual const char *__getElementName__() const noexcept = 0

virtual void __setElementName__(const char*) = 0

Name has to be a non-empty string. Top-level element can't be renamed. 'ROOT' is reserved.

virtual const char *__getElementValue__() const noexcept = 0

virtual void __setElementValue__(const char*) = 0

virtual int __getNumAttributes__() const noexcept = 0

virtual const char *__getAttributeName__(int i) const noexcept = 0

Get the name of a attribute ("" if attribute does not exist).

virtual const char *__getAttributeValue__(int i) const noexcept = 0

Get the value of a attribute ("" if attribute does not exist).

virtual const char *__getAttributeValue__(const char *name) const noexcept = 0

Get the value of a attribute of a given name ("" if attribute does not exist).

virtual void __addAttribute__(const char *name, const char *value) = 0

Add an attribute with a given name and value. If an attribute with the same name already exists, its value is replaced. Throw if name is NULL or empty.

virtual int __getNumChildrenElements__() const noexcept = 0

virtual const *FormatMetadata* &__getChildElement__(int i) const = 0

Access a child element.

Note: Adding siblings might cause a reallocation of the container and thus might make the reference unusable. Index i has to be positive and less than getNumChildrenElements() or the function will throw.

---

virtual *FormatMetadata* &__getChildElement__(int i) = 0

virtual void **addChildElement**(const char *name, const char *value) = 0

Add a child element with a given name and value.

Name has to be non-empty. Value may be empty, particularly if this element will have children. Element is added after all existing children. Use getChildElement(getNumChildrenElements()-1) to access the added element.

virtual void **clear**() noexcept = 0

Remove all children, all attributes and the value.

virtual *FormatMetadata* &**operator=**(const *FormatMetadata* &rhs) = 0

virtual const char ***getName**() const noexcept = 0

Convenience method to easily get/set the 'name' attribute. This corresponds to the ProcessNode name attribute from a CLF / CTF file or the name key of a transform in the config YAML.

virtual void **setName**(const char *name) noexcept = 0

virtual const char ***getID**() const noexcept = 0

Convenience method to easily get/set the 'id' attribute. This corresponds to the ProcessNode id attribute from a CLF/CTF file or the ColorCorrection id attribute from a CC/CCC/CDL file.

virtual void **setID**(const char *id) noexcept = 0

FormatMetadata(const *FormatMetadata* &rhs) = delete

virtual **~FormatMetadata**() = default

Do not use (needed only for pybind11).

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *FormatMetadata*&)

Constants: *FormatMetadata*

5.7.17 Global

Caching

Python

PyOpenColorIO.**ClearAllCaches**() → None

During normal usage, OpenColorIO tends to cache certain global information (such as the contents of LUTs on disk, intermediate results, etc.). Calling this function will flush all such information. The global information are related to LUT file identifications, loaded LUT file content and CDL transforms from loaded CDL files.

Under normal usage, this is not necessary, but it can be helpful in particular instances, such as designing OCIO profiles, and wanting to re-read luts without restarting.

Note: This method does not apply to instance-specific caches such as the Processor cache in a Config instance or the GPU and CPU Processor caches in a Processor instance. So in cases where you still have a Config instance after calling ClearAllCaches, you should also call the Config's clearProcessorCache method.

C++

void OpenColorIO_v2_5::**ClearAllCaches**()

During normal usage, OpenColorIO tends to cache certain global information (such as the contents of LUTs on disk, intermediate results, etc.). Calling this function will flush all such information. The global information are related to LUT file identifications, loaded LUT file content and CDL transforms from loaded CDL files.

Under normal usage, this is not necessary, but it can be helpful in particular instances, such as designing OCIO profiles, and wanting to re-read luts without restarting.

Note: This method does not apply to instance-specific caches such as the Processor cache in a Config instance or the GPU and CPU Processor caches in a Processor instance. So in cases where you still have a Config instance after calling ClearAllCaches, you should also call the Config's clearProcessorCache method.

Constants: *Caches*

Version

Python

PyOpenColorIO.**GetVersion**() → str

Get the version number for the library, as a dot-delimited string (e.g., "1.0.0").

This is also available at compile time as OCIO_VERSION_FULL_STR.

PyOpenColorIO.**GetVersionHex**() → int

Get the version number for the library, as a single 4-byte hex number (e.g., 0x01050200 for "1.5.2"), to be used for numeric comparisons.

This is also at compile time as OCIO_VERSION_HEX.

C++

const char *OpenColorIO_v2_5::**GetVersion**()

Get the version number for the library, as a dot-delimited string (e.g., "1.0.0").

This is also available at compile time as OCIO_VERSION_FULL_STR.

int OpenColorIO_v2_5::**GetVersionHex**()

Get the version number for the library, as a single 4-byte hex number (e.g., 0x01050200 for "1.5.2"), to be used for numeric comparisons.

This is also at compile time as OCIO_VERSION_HEX.

Logging

Python

PyOpenColorIO.**GetLoggingLevel**() → *PyOpenColorIO.PyOpenColorIO.LoggingLevel*

Get the global logging level.

You can override this at runtime using the OCIO_LOGGING_LEVEL environment variable. The client application that sets this should use SetLoggingLevel, and not the environment variable. The default value is INFO.

PyOpenColorIO.**SetLoggingLevel**(*level:* PyOpenColorIO.PyOpenColorIO.LoggingLevel) → None

Set the global logging level.

C++

LoggingLevel OpenColorIO_v2_5::**GetLoggingLevel**()

Get the global logging level.

You can override this at runtime using the OCIO_LOGGING_LEVEL environment variable. The client application that sets this should use SetLoggingLevel, and not the environment variable. The default value is INFO.

void OpenColorIO_v2_5::**SetLoggingLevel**(*LoggingLevel* level)

Set the global logging level.

Logging Function

Python

PyOpenColorIO.**SetLoggingFunction**(*logFunction: collections.abc.Callable[[str], None]*) → None

Set the logging function to use; otherwise, use the default (i.e. std::cerr).

Note: The logging mechanism is thread-safe.

PyOpenColorIO.**ResetToDefaultLoggingFunction**() → None

PyOpenColorIO.**LogMessage**(*level:* PyOpenColorIO.PyOpenColorIO.LoggingLevel, *message: str*) → None

Log a message using the library logging function.

C++

using OpenColorIO_v2_5::**LoggingFunction** = std::function<void(const char*)>

Define the logging function signature.

void OpenColorIO_v2_5::**SetLoggingFunction**(*LoggingFunction* logFunction)

Set the logging function to use; otherwise, use the default (i.e. std::cerr).

Note: The logging mechanism is thread-safe.

void OpenColorIO_v2_5::**ResetToDefaultLoggingFunction**()

void OpenColorIO_v2_5::**LogMessage**(*LoggingLevel* level, const char *message)

Log a message using the library logging function.

Compute Hash Function

Python

PyOpenColorIO.**SetComputeHashFunction**(*hashFunction: collections.abc.Callable[[str], str]*) → None

Set the Compute Hash Function to use; otherwise, use the default.

This is not used when using CreateFromFile with an OCIOZ archive or CreateFromConfigIOProxy.

Parameters
ComputeHashFunction –

PyOpenColorIO.**ResetComputeHashFunction**() → None

C++

using OpenColorIO_v2_5::**ComputeHashFunction** = std::function<std::string(const std::string&)>

Define Compute Hash function signature.

void OpenColorIO_v2_5::**SetComputeHashFunction**(*ComputeHashFunction* hashFunction)

Set the Compute Hash Function to use; otherwise, use the default.

This is not used when using CreateFromFile with an OCIOZ archive or CreateFromConfigIOProxy.

Parameters
ComputeHashFunction –

void OpenColorIO_v2_5::**ResetComputeHashFunction**()

Environment Variables

Python

PyOpenColorIO.**GetEnvVariable**(*name: str*) → str

Another call modifies the string obtained from a previous call as the method always uses the same memory buffer.

PyOpenColorIO.**SetEnvVariable**(*name: str*, *value: str*) → None

> **Warning:** This method is not thread safe.

PyOpenColorIO.**UnsetEnvVariable**(*name: str*) → None

> **Warning:** This method is not thread safe.

PyOpenColorIO.**IsEnvVariablePresent**(*name: str*) → bool

cpp:function:

C++

const char *OpenColorIO_v2_5::**GetEnvVariable**(const char *name)

Another call modifies the string obtained from a previous call as the method always uses the same memory buffer.

void OpenColorIO_v2_5::**SetEnvVariable**(const char *name, const char *value)

> **Warning:** This method is not thread safe.

void OpenColorIO_v2_5::**UnsetEnvVariable**(const char *name)

> **Warning:** This method is not thread safe.

bool OpenColorIO_v2_5::**IsEnvVariablePresent**(const char *name)

cpp:function::

Constants: *Environment Variables*

Casting

C++

template<class **T**, class **U**>
inline std::shared_ptr<*T*> OpenColorIO_v2_5::**DynamicPtrCast**(std::shared_ptr<*U*> const &ptr)

Conversions

Bool

Python

PyOpenColorIO.**BoolToString**(*value: bool*) → str

PyOpenColorIO.**BoolFromString**(*str: str*) → bool

C++

const char *OpenColorIO_v2_5::**BoolToString**(bool val)

bool OpenColorIO_v2_5::**BoolFromString**(const char *s)

LoggingLevel

Python

PyOpenColorIO.**LoggingLevelToString**(*level:* PyOpenColorIO.PyOpenColorIO.LoggingLevel) → str

PyOpenColorIO.**LoggingLevelFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.LoggingLevel*

C++

const char *OpenColorIO_v2_5::**LoggingLevelToString**(*LoggingLevel* level)

LoggingLevel OpenColorIO_v2_5::**LoggingLevelFromString**(const char *s)

TransformDirection

Python

PyOpenColorIO.**TransformDirectionToString**(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → str

PyOpenColorIO.**TransformDirectionFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

Will throw if string is not recognized.

PyOpenColorIO.**GetInverseTransformDirection**(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

PyOpenColorIO.**CombineTransformDirections**(*direction1:* PyOpenColorIO.PyOpenColorIO.TransformDirection, *direction2:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

C++

const char *OpenColorIO_v2_5::**TransformDirectionToString**(*TransformDirection* dir)

TransformDirection OpenColorIO_v2_5::**TransformDirectionFromString**(const char *s)

Will throw if string is not recognized.

TransformDirection OpenColorIO_v2_5::**GetInverseTransformDirection**(*TransformDirection* dir)

TransformDirection OpenColorIO_v2_5::**CombineTransformDirections**(*TransformDirection* d1, *TransformDirection* d2)

BitDepth

Python

PyOpenColorIO.**BitDepthToString**(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → str

PyOpenColorIO.**BitDepthFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.BitDepth*

PyOpenColorIO.**BitDepthIsFloat**(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → bool

PyOpenColorIO.**BitDepthToInt**(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → int

C++

const char *OpenColorIO_v2_5::**BitDepthToString**(*BitDepth* bitDepth)

BitDepth OpenColorIO_v2_5::**BitDepthFromString**(const char *s)

bool OpenColorIO_v2_5::**BitDepthIsFloat**(*BitDepth* bitDepth)

int OpenColorIO_v2_5::**BitDepthToInt**(*BitDepth* bitDepth)

Allocation

Python

PyOpenColorIO.**AllocationToString**(*allocation:* PyOpenColorIO.PyOpenColorIO.Allocation) → str

PyOpenColorIO.**AllocationFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.Allocation*

C++

const char *OpenColorIO_v2_5::**AllocationToString**(*Allocation* allocation)

Allocation OpenColorIO_v2_5::**AllocationFromString**(const char *s)

Interpolation

Python

PyOpenColorIO.**InterpolationToString**(*interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation) → str

PyOpenColorIO.**InterpolationFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.Interpolation*

C++

const char *OpenColorIO_v2_5::**InterpolationToString**(*Interpolation* interp)

Interpolation OpenColorIO_v2_5::**InterpolationFromString**(const char *s)

GpuLanguage

Python

PyOpenColorIO.**GpuLanguageToString**(*language:* PyOpenColorIO.PyOpenColorIO.GpuLanguage) → str

PyOpenColorIO.**GpuLanguageFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.GpuLanguage*

C++

const char *OpenColorIO_v2_5::**GpuLanguageToString**(*GpuLanguage* language)

GpuLanguage OpenColorIO_v2_5::**GpuLanguageFromString**(const char *s)

EnvironmentMode

Python

PyOpenColorIO.**EnvironmentModeToString**(*mode:* PyOpenColorIO.PyOpenColorIO.EnvironmentMode) → str

PyOpenColorIO.**EnvironmentModeFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.EnvironmentMode*

C++

const char *OpenColorIO_v2_5::**EnvironmentModeToString**(*EnvironmentMode* mode)

EnvironmentMode OpenColorIO_v2_5::**EnvironmentModeFromString**(const char *s)

CDLStyle

Python

PyOpenColorIO.**CDLStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.CDLStyle) → str

PyOpenColorIO.**CDLStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.CDLStyle*

C++

const char *OpenColorIO_v2_5::**CDLStyleToString**(*CDLStyle* style)

CDLStyle OpenColorIO_v2_5::**CDLStyleFromString**(const char *style)

RangeStyle

Python

PyOpenColorIO.**RangeStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.RangeStyle) → str

PyOpenColorIO.**RangeStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.RangeStyle*

C++

const char *OpenColorIO_v2_5::**RangeStyleToString**(*RangeStyle* style)

RangeStyle OpenColorIO_v2_5::**RangeStyleFromString**(const char *style)

FixedFunctionStyle

Python

PyOpenColorIO.**FixedFunctionStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.FixedFunctionStyle) → str

PyOpenColorIO.**FixedFunctionStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.FixedFunctionStyle*

C++

const char *OpenColorIO_v2_5::**FixedFunctionStyleToString**(*FixedFunctionStyle* style)

FixedFunctionStyle OpenColorIO_v2_5::**FixedFunctionStyleFromString**(const char *style)

GradingStyle

Python

PyOpenColorIO.**GradingStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → str

PyOpenColorIO.**GradingStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.GradingStyle*

C++

const char *OpenColorIO_v2_5::**GradingStyleToString**(*GradingStyle* style)

GradingStyle OpenColorIO_v2_5::**GradingStyleFromString**(const char *s)

ExposureContrastStyle

Python

PyOpenColorIO.**ExposureContrastStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.ExposureContrastStyle) → str

PyOpenColorIO.**ExposureContrastStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.ExposureContrastStyle*

C++

const char *OpenColorIO_v2_5::**ExposureContrastStyleToString**(*ExposureContrastStyle* style)

ExposureContrastStyle OpenColorIO_v2_5::**ExposureContrastStyleFromString**(const char *style)

NegativeStyle

Python

PyOpenColorIO.**NegativeStyleToString**(*style:* PyOpenColorIO.PyOpenColorIO.NegativeStyle) → str

PyOpenColorIO.**NegativeStyleFromString**(*str: str*) → *PyOpenColorIO.PyOpenColorIO.NegativeStyle*

C++

const char *OpenColorIO_v2_5::**NegativeStyleToString**(*NegativeStyle* style)

NegativeStyle OpenColorIO_v2_5::**NegativeStyleFromString**(const char *style)

5.7.18 Grading Transforms

GradingPrimaryTransform

Python

class PyOpenColorIO.**GradingPrimaryTransform**

Primary color correction controls.

This transform is for making basic color correction adjustments to an image such as brightness, contrast, or saturation.

The controls are customized for linear, logarithmic, and video color encodings. - Linear controls: Exposure, Contrast, Pivot, Offset, Saturation, Black Clip, White Clip. - Log controls: Brightness, Contrast, Pivot, Log Gamma, Saturation, Black Clip, White Clip, Black Pivot White Pivot. - Video controls : Lift, Gamma, Gain, Offset, Saturation, Black Clip, White Clip, Black Pivot White Pivot.

The controls are dynamic, so they may be adjusted even after the Transform has been included in a Processor.

GradingPrimaryTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingPrimaryTransform, values: PyOpenColorIO.PyOpenColorIO.GradingPrimary, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingPrimaryTransform.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingPrimaryTransform, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingPrimaryTransform.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getStyle() → *PyOpenColorIO.PyOpenColorIO.GradingStyle*

Adjusts the behavior of the transform for log, linear, or video color space encodings.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue() → *PyOpenColorIO.PyOpenColorIO.GradingPrimary*

isDynamic() → bool

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingPrimaryTransform only one can have dynamic parameters.

makeDynamic() → None

makeNonDynamic() → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → None

Will reset value to style's defaults if style is not the current style.

setValue(*values:* PyOpenColorIO.PyOpenColorIO.GradingPrimary) → None

Throws if value is not valid.

validate() → None

Will throw if data is not valid.

C++

class **GradingPrimaryTransform** : public *Transform*

Primary color correction controls.

This transform is for making basic color correction adjustments to an image such as brightness, contrast, or saturation.

The controls are customized for linear, logarithmic, and video color encodings.

- Linear controls: Exposure, Contrast, Pivot, Offset, Saturation, Black Clip, White Clip.
- Log controls: Brightness, Contrast, Pivot, Log Gamma, Saturation, Black Clip, White Clip, Black Pivot White Pivot.
- Video controls : Lift, Gamma, Gain, Offset, Saturation, Black Clip, White Clip, Black Pivot White Pivot.

The controls are dynamic, so they may be adjusted even after the Transform has been included in a Processor.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *GradingPrimaryTransform* &other) const noexcept = 0

Checks if this equals other.

virtual *GradingStyle* **getStyle**() const noexcept = 0

Adjusts the behavior of the transform for log, linear, or video color space encodings.

virtual void **setStyle**(*GradingStyle* style) noexcept = 0

Will reset value to style's defaults if style is not the current style.

virtual const *GradingPrimary* &**getValue**() const = 0

virtual void **setValue**(const *GradingPrimary* &values) = 0

Throws if value is not valid.

virtual bool **isDynamic**() const noexcept = 0

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingPrimaryTransform only one can have dynamic parameters.

virtual void **makeDynamic**() noexcept = 0

virtual void **makeNonDynamic**() noexcept = 0

GradingPrimaryTransform(const *GradingPrimaryTransform*&) = delete

GradingPrimaryTransform &**operator=**(const *GradingPrimaryTransform*&) = delete

virtual **~GradingPrimaryTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingPrimaryTransformRcPtr* **Create**(*GradingStyle* style)

Creates an instance of GradingPrimaryTransform.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingPrimaryTransform*&) noexcept

typedef std::shared_ptr<const *GradingPrimaryTransform*>
OpenColorIO_v2_5::**ConstGradingPrimaryTransformRcPtr**

typedef std::shared_ptr<*GradingPrimaryTransform*> OpenColorIO_v2_5::**GradingPrimaryTransformRcPtr**

GradingPrimary

Python

class PyOpenColorIO.**GradingPrimary**

Grading primary values.

NoClampBlack = -1.7976931348623157e+308

NoClampWhite = 1.7976931348623157e+308

GradingPrimary*arg0:* PyOpenColorIO.PyOpenColorIO.GradingStyle → None

property brightness

property clampBlack

property clampWhite

property contrast

property exposure

property gain

property gamma

property lift

property offset

property pivot

property pivotBlack

property pivotWhite

property saturation

validate(*arg0:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → None

The valid range for each parameter varies.

C++

struct **GradingPrimary**

Grading primary values.

Public Functions

GradingPrimary() = delete

inline explicit **GradingPrimary**(*GradingStyle* style)

void **validate**(*GradingStyle* style) const

The valid range for each parameter varies.

Public Members

GradingRGBM **m_brightness** = {0.0, 0.0, 0.0, 0.0}

GradingRGBM **m_contrast** = {1.0, 1.0, 1.0, 1.0}

GradingRGBM **m_gamma** = {1.0, 1.0, 1.0, 1.0}

GradingRGBM **m_offset** = {0.0, 0.0, 0.0, 0.0}

GradingRGBM **m_exposure** = {0.0, 0.0, 0.0, 0.0}

GradingRGBM **m_lift** = {0.0, 0.0, 0.0, 0.0}

GradingRGBM **m_gain** = {1.0, 1.0, 1.0, 1.0}

double **m_saturation** = {1.0}

double **m_pivot**

double **m_pivotBlack** = {0.0}

double **m_pivotWhite** = {1.0}

double **m_clampBlack**

double **m_clampWhite**

Public Static Functions

static double **NoClampBlack**()

static double **NoClampWhite**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingPrimary*&)

GradingRGBM

Python

class PyOpenColorIO.**GradingRGBM**

Used by the grading transforms to hold the red, green, blue, and master components of a single parameter. The master component affects all three channels (RGB).

GradingRGBM(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBM) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBM, red: typing.SupportsFloat, green: typing.SupportsFloat, blue: typing.SupportsFloat, master: typing.SupportsFloat) -> None

property blue

property green

property master

property red

C++

struct **GradingRGBM**

Used by the grading transforms to hold the red, green, blue, and master components of a single parameter. The master component affects all three channels (RGB).

Public Functions

GradingRGBM() = default

inline **GradingRGBM**(double red, double green, double blue, double master)

inline **GradingRGBM**(const double (&rgbm)[4])

Public Members

double **m_red** = {0.}

double **m_green** = {0.}

double **m_blue** = {0.}

double **m_master** = {0.}

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingRGBM*&)

GradingRGBCurveTransform

Python

class PyOpenColorIO.**GradingRGBCurveTransform**

RGB curve color correction controls.

This transform allows for modifying tone reproduction via B-spline curves.

There is an R, G, and B curve followed by a Master curve (that applies to R, G, and B). Each curve is specified via the x and y coordinates of its control points. A monotonic spline is fit to the control points. The x coordinates must be non-decreasing. When the grading style is linear, the units for the control points are photographic stops relative to 0.18.

The control points are dynamic, so they may be adjusted even after the Transform is included in a Processor.

GradingRGBCurveTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBCurveTransform, values: PyOpenColorIO.PyOpenColorIO.GradingRGBCurve, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingPrimaryTransform.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBCurveTransform, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingPrimaryTransform.

getBypassLinToLog() → bool

The scene-linear grading style applies a lin-to-log transform to the pixel values before going through the curve. However, in some cases (e.g. drawing curves in a UI) it may be useful to bypass the lin-to-log. Default value is false.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getSlope(*channel:* PyOpenColorIO.PyOpenColorIO.RGBCurveType, *index: SupportsInt*) → float

It is possible to provide a desired slope value for each control point. The number of slopes is always the same as the number of control points and so the control points must be set before setting the slopes. The slopes are primarily intended for use by config authors looking to match a specific shape with as few control points as possible, they are not intended to be exposed to a user interface for direct manipulation. When a curve is being generated for creative purposes it is better to let OCIO calculate the slopes automatically.

getStyle() → *PyOpenColorIO.PyOpenColorIO.GradingStyle*

Adjusts the behavior of the transform for log, linear, or video color space encodings.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue() → *PyOpenColorIO.PyOpenColorIO.GradingRGBCurve*

isDynamic() → bool

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingRGBCurveTransform only one can have dynamic parameters.

makeDynamic() → None

makeNonDynamic() → None

setBypassLinToLog(*bypass: bool*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setSlope(*channel:* PyOpenColorIO.PyOpenColorIO.RGBCurveType, *index: SupportsInt*, *slope: SupportsFloat*) → None

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → None

Will reset value to style's defaults if style is not the current style.

setValue(*values:* PyOpenColorIO.PyOpenColorIO.GradingRGBCurve) → None

Throws if value is not valid.

slopesAreDefault(*channel:* PyOpenColorIO.PyOpenColorIO.RGBCurveType) → bool

validate() → None

Will throw if data is not valid.

C++

class **GradingRGBCurveTransform** : public *Transform*

RGB curve color correction controls.

This transform allows for modifying tone reproduction via B-spline curves.

There is an R, G, and B curve followed by a Master curve (that applies to R, G, and B). Each curve is specified via the x and y coordinates of its control points. A monotonic spline is fit to the control points. The x coordinates must be non-decreasing. When the grading style is linear, the units for the control points are photographic stops relative to 0.18.

The control points are dynamic, so they may be adjusted even after the Transform is included in a Processor.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *GradingRGBCurveTransform* &other) const noexcept = 0

Checks if this equals other.

virtual *GradingStyle* **getStyle**() const noexcept = 0

Adjusts the behavior of the transform for log, linear, or video color space encodings.

virtual void **setStyle**(*GradingStyle* style) noexcept = 0

Will reset value to style's defaults if style is not the current style.

virtual const *ConstGradingRGBCurveRcPtr* **getValue**() const = 0

virtual void **setValue**(const *ConstGradingRGBCurveRcPtr* &values) = 0

Throws if value is not valid.

virtual float **getSlope**(*RGBCurveType* c, size_t index) const = 0

It is possible to provide a desired slope value for each control point. The number of slopes is always the same as the number of control points and so the control points must be set before setting the slopes. The slopes are primarily intended for use by config authors looking to match a specific shape with as few control points as possible, they are not intended to be exposed to a user interface for direct manipulation. When a curve is being generated for creative purposes it is better to let OCIO calculate the slopes automatically.

virtual void **setSlope**(*RGBCurveType* c, size_t index, float slope) = 0

virtual bool **slopesAreDefault**(*RGBCurveType* c) const = 0

virtual bool **getBypassLinToLog**() const = 0

The scene-linear grading style applies a lin-to-log transform to the pixel values before going through the curve. However, in some cases (e.g. drawing curves in a UI) it may be useful to bypass the lin-to-log. Default value is false.

virtual void **setBypassLinToLog**(bool bypass) = 0

virtual bool **isDynamic**() const noexcept = 0

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingRGBCurveTransform only one can have dynamic parameters.

virtual void **makeDynamic**() noexcept = 0

virtual void **makeNonDynamic**() noexcept = 0

GradingRGBCurveTransform(const *GradingRGBCurveTransform*&) = delete

GradingRGBCurveTransform &**operator=**(const *GradingRGBCurveTransform*&) = delete

virtual **~GradingRGBCurveTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingRGBCurveTransformRcPtr* **Create**(*GradingStyle* style)

Creates an instance of GradingPrimaryTransform.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingRGBCurveTransform*&) noexcept

typedef std::shared_ptr<const *GradingRGBCurveTransform*>
OpenColorIO_v2_5::**ConstGradingRGBCurveTransformRcPtr**

typedef std::shared_ptr<*GradingRGBCurveTransform*>
OpenColorIO_v2_5::**GradingRGBCurveTransformRcPtr**

GradingRGBCurve

Python

class PyOpenColorIO.**GradingRGBCurve**

A set of red, green, blue and master curves. It is used by RGBCurveTransform and can be used as a dynamic property (see DynamicPropertyGradingRGBCurve).

GradingRGBCurve(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBCurve, style: PyOpenColorIO.PyOpenColorIO.GradingStyle) -> None

Do not use (needed only for pybind11).

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBCurve, red: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve = <control_points=[<x=0, y=0><x=0.5, y=0.5><x=1, y=1>]>, green: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve = <control_points=[<x=0, y=0><x=0.5, y=0.5><x=1, y=1>]>, blue: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve = <control_points=[<x=0, y=0><x=0.5, y=0.5><x=1, y=1>]>, master: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve = <control_points=[<x=0, y=0><x=0.5, y=0.5><x=1, y=1>]>) -> None

property blue

property green

isIdentity() → bool

property master

property red

validate() → None

C++

class **GradingRGBCurve**

A set of red, green, blue and master curves. It is used by RGBCurveTransform and can be used as a dynamic property (see DynamicPropertyGradingRGBCurve).

Public Functions

virtual *GradingRGBCurveRcPtr* **createEditableCopy**() const = 0

virtual void **validate**() const = 0

virtual bool **isIdentity**() const = 0

virtual *ConstGradingBSplineCurveRcPtr* **getCurve**(*RGBCurveType* c) const = 0

virtual *GradingBSplineCurveRcPtr* **getCurve**(*RGBCurveType* c) = 0

virtual **~GradingRGBCurve**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingRGBCurveRcPtr* **Create**(*GradingStyle* style)

Create a GradingRGBCurve. (The style argument is not part of the object, it is simply used to initialize the proper default curves.)

static *GradingRGBCurveRcPtr* **Create**(const *ConstGradingRGBCurveRcPtr* &rhs)

static *GradingRGBCurveRcPtr* **Create**(const *ConstGradingBSplineCurveRcPtr* &red, const *ConstGradingBSplineCurveRcPtr* &green, const *ConstGradingBSplineCurveRcPtr* &blue, const *ConstGradingBSplineCurveRcPtr* &master)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingRGBCurve*&)

typedef std::shared_ptr<const *GradingRGBCurve*> OpenColorIO_v2_5::**ConstGradingRGBCurveRcPtr**

typedef std::shared_ptr<*GradingRGBCurve*> OpenColorIO_v2_5::**GradingRGBCurveRcPtr**

GradingHueCurveTransform

Python

class PyOpenColorIO.**GradingHueCurveTransform**

Hue curve color correction controls.

This transform provides eight spline curves to make the following adjustments:

- Hue-Hue: Map input hue to output hue (where a diagonal line is the identity).
- Hue-Sat: Adjust saturation as a function of hue (a value of 1.0 is the identity).

- Hue-Lum: Adjust luma as a function of hue (a value of 1.0 is the identity).
- Lum-Sat: Adjust saturation as a function of luma (a value of 1.0 is the identity).
- Sat-Sat: Adjust saturation as a function of saturation (a diagonal is the identity).
- Lum-Lum: Adjust luma as a function of luma, maintaining hue & sat (diagonal is identity).
- Sat-Lum: Adjust luma as a function of saturation (a value of 1.0 is the identity).
- Hue-FX : Map input hue to delta output hue (a value of 0.0 is the identity).

The algorithm is different for scene-linear, logarithmic, and video color spaces, so initialize the style argument appropriately before setting the curves.

An RGB-to-HSY FixedFunction is used to convert RGB into a hue, saturation, luma color space. However, there is an option to bypass that conversion to use an outboard transform.

Like the GradingRGBCurveTransform, the curves are defined by the x and y coordinates of a set of control points. A spline will be fit to the control points. Monotonicity is preserved for curves where the diagonal is the identity. For curves that take luma as input, if the style is scene-linear, the units are in photographic stops relative to 0.18. For log and video, the luma is scaled the same as the input RGB.

The hue variable is [0,1] and is periodic. For example, -0.2, 0.8, and 1.8 are equivalent. The domain of the curves is [0,1] and control points outside that domain are mapped into it. A hue of 0 or 1 corresponds to a magenta hue.

The control points are dynamic, so they may be adjusted even after the Transform is included in a Processor. However, creating a curve or setting the parameters will call the GradingBSplineCurveImpl::validate function, which will throw an exception if the control points do not meet certain requirements, for example that the X-coordinates are non-decreasing Please review that function for details on the validation. Applications that provide a UI to edit curves must ensure that they prevent users from creating control points that are not valid.

The transform is invertible as long as the curves allow it. For example, if saturation is mapped to zero, obviously that cannot be resaturated. Care should be taken with the Hue-FX curve because it is possible to fold hues over on themselves, which also cannot be inverted. In most cases the Hue-FX curve is not necessary since the Hue-Hue curve provides similar functionality with the added benefit of being strictly invertible.

GradingHueCurveTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingHueCurveTransform, values: PyOpenColorIO.PyOpenColorIO.GradingHueCurve, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingHueCurveTransform.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingHueCurveTransform, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingHueCurveTransform.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getRGBToHSY() → PyOpenColorIO.PyOpenColorIO.HSYTransformStyle

By default, the input is transformed into HSY space to apply the hue curves and then the result is transformed back to RGB. However, this may be set to HSY_TRANSFORM_NONE to bypass this in order to use other hue/sat/luma type transforms applied separately before and after this transform.

getSlope(*curve: PyOpenColorIO.PyOpenColorIO.HueCurveType*, *index: SupportsInt*) → float

It is possible to provide a desired slope value for each control point. The number of slopes is always the same as the number of control points and so the control points must be set before setting the slopes. The slopes are primarily intended for use by config authors looking to match a specific shape with as few control points as possible, they are not intended to be exposed to a user interface for direct manipulation. When a curve is being generated for creative purposes it is better to let OCIO calculate the slopes automatically.

getStyle() → *PyOpenColorIO.PyOpenColorIO.GradingStyle*

Adjusts the behavior of the transform for log, linear, or video color space encodings.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue() → *PyOpenColorIO.PyOpenColorIO.GradingHueCurve*

isDynamic() → bool

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingHueCurveTransform only one can have dynamic parameters.

makeDynamic() → None

makeNonDynamic() → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setRGBToHSY(*style: PyOpenColorIO.PyOpenColorIO.HSYTransformStyle*) → None

setSlope(*curve: PyOpenColorIO.PyOpenColorIO.HueCurveType*, *index: SupportsInt*, *slope: SupportsFloat*) → None

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → None

Will reset value to style's defaults if style is not the current style.

setValue(*values:* PyOpenColorIO.PyOpenColorIO.GradingHueCurve) → None

Throws if value is not valid.

slopesAreDefault(*curve: PyOpenColorIO.PyOpenColorIO.HueCurveType*) → bool

validate() → None

Will throw if data is not valid.

C++

class **GradingHueCurveTransform** : public *Transform*

Hue curve color correction controls.

This transform provides eight spline curves to make the following adjustments:

- Hue-Hue: Map input hue to output hue (where a diagonal line is the identity).
- Hue-Sat: Adjust saturation as a function of hue (a value of 1.0 is the identity).
- Hue-Lum: Adjust luma as a function of hue (a value of 1.0 is the identity).
- Lum-Sat: Adjust saturation as a function of luma (a value of 1.0 is the identity).
- Sat-Sat: Adjust saturation as a function of saturation (a diagonal is the identity).
- Lum-Lum: Adjust luma as a function of luma, maintaining hue & sat (diagonal is identity).

- Sat-Lum: Adjust luma as a function of saturation (a value of 1.0 is the identity).
- Hue-FX : Map input hue to delta output hue (a value of 0.0 is the identity).

The algorithm is different for scene-linear, logarithmic, and video color spaces, so initialize the style argument appropriately before setting the curves.

An RGB-to-HSY FixedFunction is used to convert RGB into a hue, saturation, luma color space. However, there is an option to bypass that conversion to use an outboard transform.

Like the GradingRGBCurveTransform, the curves are defined by the x and y coordinates of a set of control points. A spline will be fit to the control points. Monotonicity is preserved for curves where the diagonal is the identity. For curves that take luma as input, if the style is scene-linear, the units are in photographic stops relative to 0.18. For log and video, the luma is scaled the same as the input RGB.

The hue variable is [0,1] and is periodic. For example, -0.2, 0.8, and 1.8 are equivalent. The domain of the curves is [0,1] and control points outside that domain are mapped into it. A hue of 0 or 1 corresponds to a magenta hue.

The control points are dynamic, so they may be adjusted even after the Transform is included in a Processor. However, creating a curve or setting the parameters will call the GradingBSplineCurveImpl::validate function, which will throw an exception if the control points do not meet certain requirements, for example that the X-coordinates are non-decreasing Please review that function for details on the validation. Applications that provide a UI to edit curves must ensure that they prevent users from creating control points that are not valid.

The transform is invertible as long as the curves allow it. For example, if saturation is mapped to zero, obviously that cannot be resaturated. Care should be taken with the Hue-FX curve because it is possible to fold hues over on themselves, which also cannot be inverted. In most cases the Hue-FX curve is not necessary since the Hue-Hue curve provides similar functionality with the added benefit of being strictly invertible.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *GradingHueCurveTransform* &other) const noexcept = 0

Checks if this equals other.

virtual *GradingStyle* **getStyle**() const noexcept = 0

Adjusts the behavior of the transform for log, linear, or video color space encodings.

virtual void **setStyle**(*GradingStyle* style) noexcept = 0

Will reset value to style's defaults if style is not the current style.

virtual const *ConstGradingHueCurveRcPtr* **getValue**() const = 0

virtual void **setValue**(const *ConstGradingHueCurveRcPtr* &value) = 0

Throws if value is not valid.

virtual float **getSlope**(HueCurveType c, size_t index) const = 0

It is possible to provide a desired slope value for each control point. The number of slopes is always the same as the number of control points and so the control points must be set before setting the slopes. The slopes are primarily intended for use by config authors looking to match a specific shape with as few control points as possible, they are not intended to be exposed to a user interface for direct manipulation. When a curve is being generated for creative purposes it is better to let OCIO calculate the slopes automatically.

virtual void **setSlope**(HueCurveType c, size_t index, float slope) = 0

virtual bool **slopesAreDefault**(HueCurveType c) const = 0

virtual HSYTransformStyle **getRGBToHSY**() const = 0

By default, the input is transformed into HSY space to apply the hue curves and then the result is transformed back to RGB. However, this may be set to HSY_TRANSFORM_NONE to bypass this in order to use other hue/sat/luma type transforms applied separately before and after this transform.

virtual void **setRGBToHSY**(HSYTransformStyle style) = 0

virtual bool **isDynamic**() const noexcept = 0

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingHueCurveTransform only one can have dynamic parameters.

virtual void **makeDynamic**() noexcept = 0

virtual void **makeNonDynamic**() noexcept = 0

GradingHueCurveTransform(const *GradingHueCurveTransform*&) = delete

GradingHueCurveTransform &**operator=**(const *GradingHueCurveTransform*&) = delete

virtual **~GradingHueCurveTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingHueCurveTransformRcPtr* **Create**(*GradingStyle* style)

Creates an instance of GradingHueCurveTransform.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingHueCurveTransform*&) noexcept

typedef std::shared_ptr<const *GradingHueCurveTransform*>
OpenColorIO_v2_5::**ConstGradingHueCurveTransformRcPtr**

typedef std::shared_ptr<*GradingHueCurveTransform*> OpenColorIO_v2_5::**GradingHueCurveTransformRcPtr**

GradingHueCurve

Python

class PyOpenColorIO.**GradingHueCurve**

A set of HUE/SAT/LUM curves. It is used by GradingHueCurveTransform and can be used as a dynamic property (see DynamicPropertyGradingHueCurve).

GradingHueCurve(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingHueCurve, style: PyOpenColorIO.PyOpenColorIO.GradingStyle) -> None

Do not use (needed only for pybind11).

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingHueCurve, arg0: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg1: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg2: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg3: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg4: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg5: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg6: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg7: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve) -> None

getDrawCurveOnly() → bool

Enable drawCurveOnly mode to return the output value of a spline curve without any of the other associated processing of the RGB values. This is useful when the curves need to be graphed independently in a user interface. To use this, set the curve parameters on the Hue-Sat curve. The R, G, and B values will be sent through that curve with the interpretation that they are the input axis to the curve (which would be hue, sat, or luma) rather than RGB. This mode does not apply the RGB-to-HSY or Lin-to-Log, so for scene-linear curves the luma values are interpreted as already being in the logarithmic (f-stop) space. The forward curve evaluation is done regardless of the transform direction.

property hue_fx

property hue_hue

property hue_lum

property hue_sat

isIdentity() → bool

property lum_lum

property lum_sat

property sat_lum

property sat_sat

setDrawCurveOnly(*drawcurveonly: bool*) → None

validate() → None

C++

class **GradingHueCurve**

A set of HUE/SAT/LUM curves. It is used by GradingHueCurveTransform and can be used as a dynamic property (see DynamicPropertyGradingHueCurve).

Public Functions

virtual *GradingHueCurveRcPtr* **createEditableCopy**() const = 0

virtual void **validate**() const = 0

virtual bool **isIdentity**() const = 0

virtual bool **getDrawCurveOnly**() const = 0

Enable drawCurveOnly mode to return the output value of a spline curve without any of the other associated processing of the RGB values. This is useful when the curves need to be graphed independently in a user interface. To use this, set the curve parameters on the Hue-Sat curve. The R, G, and B values will be sent through that curve with the interpretation that they are the input axis to the curve (which would be hue, sat, or luma) rather than RGB. This mode does not apply the RGB-to-HSY or Lin-to-Log, so for scene-linear curves the luma values are interpreted as already being in the logarithmic (f-stop) space. The forward curve evaluation is done regardless of the transform direction.

virtual void **setDrawCurveOnly**(bool drawCurveOnly) = 0

virtual *ConstGradingBSplineCurveRcPtr* **getCurve**(HueCurveType c) const = 0

virtual *GradingBSplineCurveRcPtr* **getCurve**(HueCurveType c) = 0

virtual **~GradingHueCurve**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingHueCurveRcPtr* **Create**(*GradingStyle* style)

Create a GradingHueCurve. (The style argument is not part of the object, it is simply used to initialize the proper default curves.)

static *GradingHueCurveRcPtr* **Create**(const *ConstGradingHueCurveRcPtr* &rhs)

static *GradingHueCurveRcPtr* **Create**(*ConstGradingBSplineCurveRcPtr* hueHue,
ConstGradingBSplineCurveRcPtr hueSat,
ConstGradingBSplineCurveRcPtr hueLum,
ConstGradingBSplineCurveRcPtr lumSat,
ConstGradingBSplineCurveRcPtr satSat,
ConstGradingBSplineCurveRcPtr lumLum,
ConstGradingBSplineCurveRcPtr satLum,
ConstGradingBSplineCurveRcPtr hueFx)

static BSplineType **GetBSplineTypeForHueCurveType**(HueCurveType curveType)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingHueCurve*&)

typedef std::shared_ptr<const *GradingHueCurve*> OpenColorIO_v2_5::**ConstGradingHueCurveRcPtr**

typedef std::shared_ptr<*GradingHueCurve*> OpenColorIO_v2_5::**GradingHueCurveRcPtr**

GradingControlPoint

Python

class PyOpenColorIO.**GradingControlPoint**

2D control point used by GradingBSplineCurve.

GradingControlPoint(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingControlPoint) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingControlPoint, x: typing.SupportsFloat = 0.0, y: typing.SupportsFloat = 0.0) -> None

property x

property y

C++

struct **GradingControlPoint**

2D control point used by GradingBSplineCurve.

Public Functions

GradingControlPoint() = default

inline **GradingControlPoint**(float x, float y)

Public Members

float **m_x** = {0.f}

float **m_y** = {0.f}

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingControlPoint*&)

GradingBSplineCurve

Python

class PyOpenColorIO.**GradingBSplineCurve**

A BSpline curve defined with GradingControlPoint.

GradingBSplineCurve(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, size: typing.SupportsInt) -> None

Create a BSpline curve with a specified number of control points.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, size: typing.SupportsInt, huecurvetype: PyOpenColorIO.PyOpenColorIO.HueCurveType) -> None

Create a BSpline curve with a specified number of control points.

3. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, size: typing.SupportsInt, splinetype: PyOpenColorIO.PyOpenColorIO.BSplineType) -> None

Create a BSpline curve with a specified number of control points.

4. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg0: collections.abc.Sequence[typing.SupportsFloat]) -> None

5. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve, arg0: collections.abc.Sequence[typing.SupportsFloat], arg1: PyOpenColorIO.PyOpenColorIO.HueCurveType) -> None

getControlPoints() → *PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator*

getSlopes() → list[float]

getSplineType() → PyOpenColorIO.PyOpenColorIO.BSplineType

setNumControlPoints(*size: SupportsInt*) → None

setSlopes(*arg0: collections.abc.Sequence[SupportsFloat]*) → None

setSplineType(*splinetype: PyOpenColorIO.PyOpenColorIO.BSplineType*) → None

slopesAreDefault() → bool

validate() → None

class PyOpenColorIO.GradingBSplineCurve.**GradingControlPointIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.GradingControlPoint*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.GradingControlPoint*

self [*arg0: SupportsInt*] = *arg1: PyOpenColorIO.PyOpenColorIO.GradingControlPoint* → None

C++

class **GradingBSplineCurve**

A BSpline curve defined with GradingControlPoint.

Public Functions

virtual *GradingBSplineCurveRcPtr* **createEditableCopy**() const = 0

virtual size_t **getNumControlPoints**() const noexcept = 0

Get the number of ControlPoint objects (and the number of slopes).

virtual void **setNumControlPoints**(size_t size) = 0

virtual const *GradingControlPoint* &**getControlPoint**(size_t index) const = 0

virtual *GradingControlPoint* &**getControlPoint**(size_t index) = 0

virtual float **getSlope**(size_t index) const = 0

virtual void **setSlope**(size_t index, float slope) = 0

virtual bool **slopesAreDefault**() const = 0

virtual void **validate**() const = 0

virtual BSplineType **getSplineType**() const = 0

virtual void **setSplineType**(BSplineType splineType) = 0

GradingBSplineCurve(const *GradingBSplineCurve*&) = delete

GradingBSplineCurve &**operator=**(const *GradingBSplineCurve*&) = delete

virtual **~GradingBSplineCurve**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingBSplineCurveRcPtr* **Create**(size_t size)

Create a BSpline curve with a specified number of control points.

static *GradingBSplineCurveRcPtr* **Create**(size_t size, BSplineType splineType)

static *GradingBSplineCurveRcPtr* **Create**(size_t size, HueCurveType curveType)

static *GradingBSplineCurveRcPtr* **Create**(std::initializer_list<*GradingControlPoint*> values)

Create a BSpline curve with a list of control points.

static *GradingBSplineCurveRcPtr* **Create**(std::initializer_list<*GradingControlPoint*> values, BSplineType splineType)

static *GradingBSplineCurveRcPtr* **Create**(std::initializer_list<*GradingControlPoint*> values, HueCurveType curveType)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingBSplineCurve*&)

typedef std::shared_ptr<const *GradingBSplineCurve*> OpenColorIO_v2_5::**ConstGradingBSplineCurveRcPtr**

typedef std::shared_ptr<*GradingBSplineCurve*> OpenColorIO_v2_5::**GradingBSplineCurveRcPtr**

GradingToneTransform

Python

class PyOpenColorIO.**GradingToneTransform**

Tonal color correction controls.

This transform is for making fine adjustments to tone reproduction in specific tonal ranges.

There are five tonal controls and each one has two parameters to control its range: - Blacks (start, width) - Shadows(start, pivot) - Midtones(center, width) - Highlights(start, pivot) - Whites(start, width)

The transform has three styles that adjust the response and default ranges for linear, logarithimic, and video color encodings. The defaults vary based on the style. When the style is linear, the units for start/width/etc. are photographic stops relative to 0.18.

Each control allows R, G, B adjustments and a Master adjustment.

There is also an S-contrast control for imparting an S-shape curve.

The controls are dynamic, so they may be adjusted even after the Transform has been included in a Processor.

GradingToneTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingToneTransform, values: PyOpenColorIO.PyOpenColorIO.GradingTone, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingToneTransform.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingToneTransform, style: PyOpenColorIO.PyOpenColorIO.GradingStyle = <GradingStyle.GRADING_LOG: 0>, dynamic: bool = False, dir: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of GradingToneTransform.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getStyle() → *PyOpenColorIO.PyOpenColorIO.GradingStyle*

Adjusts the behavior of the transform for log, linear, or video color space encodings.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue() → *PyOpenColorIO.PyOpenColorIO.GradingTone*

isDynamic() → bool

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingToneTransform only one can have dynamic parameters.

makeDynamic() → None

makeNonDynamic() → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.GradingStyle) → None

Will reset value to style's defaults if style is not the current style.

setValue(*values:* PyOpenColorIO.PyOpenColorIO.GradingTone) → None

validate() → None

Will throw if data is not valid.

C++

class **GradingToneTransform** : public *Transform*

Tonal color correction controls.

This transform is for making fine adjustments to tone reproduction in specific tonal ranges.

There are five tonal controls and each one has two parameters to control its range:

- Blacks (start, width)
- Shadows(start, pivot)

- Midtones(center, width)
- Highlights(start, pivot)
- Whites(start, width)

The transform has three styles that adjust the response and default ranges for linear, logarithimic, and video color encodings. The defaults vary based on the style. When the style is linear, the units for start/width/etc. are photographic stops relative to 0.18.

Each control allows R, G, B adjustments and a Master adjustment.

There is also an S-contrast control for imparting an S-shape curve.

The controls are dynamic, so they may be adjusted even after the Transform has been included in a Processor.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *GradingToneTransform* &other) const noexcept = 0

virtual *GradingStyle* **getStyle**() const noexcept = 0

Adjusts the behavior of the transform for log, linear, or video color space encodings.

virtual void **setStyle**(*GradingStyle* style) noexcept = 0

Will reset value to style's defaults if style is not the current style.

virtual const *GradingTone* &**getValue**() const = 0

virtual void **setValue**(const *GradingTone* &values) = 0

virtual bool **isDynamic**() const noexcept = 0

Parameters can be made dynamic so the values can be changed through the CPU or GPU processor, but if there are several GradingToneTransform only one can have dynamic parameters.

virtual void **makeDynamic**() noexcept = 0

virtual void **makeNonDynamic**() noexcept = 0

GradingToneTransform(const *GradingToneTransform*&) = delete

GradingToneTransform &**operator=**(const *GradingToneTransform*&) = delete

virtual **~GradingToneTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GradingToneTransformRcPtr* **Create**(*GradingStyle* style)

Creates an instance of GradingToneTransform.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingToneTransform*&) noexcept

typedef std::shared_ptr<const *GradingToneTransform*>
OpenColorIO_v2_5::**ConstGradingToneTransformRcPtr**

typedef std::shared_ptr<*GradingToneTransform*> OpenColorIO_v2_5::**GradingToneTransformRcPtr**

GradingTone

Python

class PyOpenColorIO.**GradingTone**

Grading tone values.

GradingTone*arg0:* PyOpenColorIO.PyOpenColorIO.GradingStyle → None

property blacks

property highlights

property midtones

property scontrast

property shadows

validate() → None

The valid range for each parameter varies. The client is expected to enforce these bounds in the UI.

property whites

C++

struct **GradingTone**

Grading tone values.

Public Functions

GradingTone() = delete

inline explicit **GradingTone**(*GradingStyle* style)

void **validate**() const

The valid range for each parameter varies. The client is expected to enforce these bounds in the UI.

Public Members

GradingRGBMSW **m_blacks**

GradingRGBMSW **m_shadows**

GradingRGBMSW **m_midtones**

GradingRGBMSW **m_highlights**

GradingRGBMSW **m_whites**

double **m_scontrast** = {1.0}

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingTone*&)

GradingRGBMSW

Python

class PyOpenColorIO.**GradingRGBMSW**

Used by the grading tone transforms to hold the red, green, blue, master, start, and width components of a single parameter. The master component affects all three channels (RGB). The start and width components control the range of tones affected. Although this struct simply uses "start" and "width" for all the range values, the actual user-facing name changes based on the parameter.

GradingRGBMSW(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBMSW) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBMSW, red: typing.SupportsFloat, green: typing.SupportsFloat, blue: typing.SupportsFloat, master: typing.SupportsFloat, start: typing.SupportsFloat, width: typing.SupportsFloat) -> None
3. __init__(self: PyOpenColorIO.PyOpenColorIO.GradingRGBMSW, start: typing.SupportsFloat, width: typing.SupportsFloat) -> None

property blue

property green

property master

property red

property start

property width

C++

struct **GradingRGBMSW**

Used by the grading tone transforms to hold the red, green, blue, master, start, and width components of a single parameter. The master component affects all three channels (RGB). The start and width components control the range of tones affected. Although this struct simply uses "start" and "width" for all the range values, the actual user-facing name changes based on the parameter.

Public Functions

GradingRGBMSW() = default

inline **GradingRGBMSW**(double red, double green, double blue, double master, double start, double width)

inline **GradingRGBMSW**(const double (&rgbmsw)[6])

inline **GradingRGBMSW**(double start, double width)

Public Members

double **m_red** = {1.}

double **m_green** = {1.}

double **m_blue** = {1.}

double **m_master** = {1.}

double **m_start** = {0.}

double **m_width** = {1.}

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GradingRGBMSW*&)

5.7.19 ImageDesc

Python

class PyOpenColorIO.**ImageDesc**

This is a light-weight wrapper around an image, that provides a context for pixel access. This does NOT claim ownership of the pixels or copy image data.

ImageDesc → None

getBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Get the bit-depth.

getHeight() → int

Get the height to process (where y position starts at 0 and ends at height-1).

getWidth() → int

Get the width to process (where x position starts at 0 and ends at width-1).

getXStrideBytes() → int

Get the step in bytes to find the same color channel of the next pixel.

getYStrideBytes() → int

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

isFloat() → bool

Is the image buffer 32-bit float?

isRGBAPacked() → bool

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

PyOpenColorIO.**AutoStride: int**

C++

class **ImageDesc**

This is a light-weight wrapper around an image, that provides a context for pixel access. This does NOT claim ownership of the pixels or copy image data.

Subclassed by PackedImageDesc, PlanarImageDesc

Public Functions

ImageDesc()

virtual **~ImageDesc**()

virtual void ***getRData**() const = 0

Get a pointer to the red channel of the first pixel.

virtual void ***getGData**() const = 0

Get a pointer to the green channel of the first pixel.

virtual void ***getBData**() const = 0

Get a pointer to the blue channel of the first pixel.

virtual void ***getAData**() const = 0

Get a pointer to the alpha channel of the first pixel or null as alpha channel is optional.

virtual *BitDepth* **getBitDepth**() const = 0

Get the bit-depth.

virtual long **getWidth**() const = 0

Get the width to process (where x position starts at 0 and ends at width-1).

virtual long **getHeight**() const = 0

Get the height to process (where y position starts at 0 and ends at height-1).

virtual ptrdiff_t **getXStrideBytes**() const = 0

Get the step in bytes to find the same color channel of the next pixel.

virtual ptrdiff_t **getYStrideBytes**() const = 0

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

virtual bool **isRGBAPacked**() const = 0

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

virtual bool **isFloat**() const = 0

Is the image buffer 32-bit float?

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ImageDesc*&)

typedef std::shared_ptr<const *ImageDesc*> OpenColorIO_v2_5::**ConstImageDescRcPtr**

typedef std::shared_ptr<*ImageDesc*> OpenColorIO_v2_5::**ImageDescRcPtr**

const ptrdiff_t OpenColorIO_v2_5::**AutoStride** = std::numeric_limits<ptrdiff_t>::min()

PackedImageDesc

Python

class PyOpenColorIO.**PackedImageDesc**

All the constructors expect a pointer to packed image data (such as rgbrgbrgb or rgbargbargba) starting at the first color channel of the first pixel to process (which does not need to be the first pixel of the image). The number of channels must be greater than or equal to 3. If a 4th channel is specified, it is assumed to be alpha information. Channels > 4 will be ignored.

Note: The methods assume the CPUProcessor bit-depth type for the data pointer.

PackedImageDesc(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.PackedImageDesc, data: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, numChannels: typing.SupportsInt) -> None

---

Note: numChannels must be 3 (RGB) or 4 (RGBA).

---

2. __init__(self: PyOpenColorIO.PyOpenColorIO.PackedImageDesc, data: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, numChannels: typing.SupportsInt, bitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, chanStrideBytes: typing.SupportsInt, xStrideBytes: typing.SupportsInt, yStrideBytes: typing.SupportsInt) -> None

---

Note: numChannels must be 3 (RGB) or 4 (RGBA).

---

3. __init__(self: PyOpenColorIO.PyOpenColorIO.PackedImageDesc, data: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, chanOrder: PyOpenColorIO.PyOpenColorIO.ChannelOrdering) -> None

4. __init__(self: PyOpenColorIO.PyOpenColorIO.PackedImageDesc, data: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, chanOrder: PyOpenColorIO.PyOpenColorIO.ChannelOrdering, bitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, chanStrideBytes: typing.SupportsInt, xStrideBytes: typing.SupportsInt, yStrideBytes: typing.SupportsInt) -> None

getBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Get the bit-depth.

getChanStrideBytes() → int

getChannelOrder() → *PyOpenColorIO.PyOpenColorIO.ChannelOrdering*

Get the channel ordering of all the pixels.

getData() → numpy.ndarray

Get a pointer to the first color channel of the first pixel.

getHeight() → int

Get the height to process (where y position starts at 0 and ends at height-1).

getNumChannels() → int

getWidth() → int

Get the width to process (where x position starts at 0 and ends at width-1).

getXStrideBytes() → int

Get the step in bytes to find the same color channel of the next pixel.

getYStrideBytes() → int

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

isFloat() → bool

Is the image buffer 32-bit float?

isRGBAPacked() → bool

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

C++

class **PackedImageDesc** : public *ImageDesc*

All the constructors expect a pointer to packed image data (such as rgbrgbrgb or rgbargbargba) starting at the first color channel of the first pixel to process (which does not need to be the first pixel of the image). The number of channels must be greater than or equal to 3. If a 4th channel is specified, it is assumed to be alpha information. Channels > 4 will be ignored.

Note: The methods assume the CPUProcessor bit-depth type for the data pointer.

Public Functions

PackedImageDesc(void *data, long width, long height, long numChannels)

> **Note:** numChannels must be 3 (RGB) or 4 (RGBA).

PackedImageDesc(void *data, long width, long height, long numChannels, *BitDepth* bitDepth, ptrdiff_t chanStrideBytes, ptrdiff_t xStrideBytes, ptrdiff_t yStrideBytes)

> **Note:** numChannels must be 3 (RGB) or 4 (RGBA).

PackedImageDesc(void *data, long width, long height, *ChannelOrdering* chanOrder)

PackedImageDesc(void *data, long width, long height, *ChannelOrdering* chanOrder, *BitDepth* bitDepth, ptrdiff_t chanStrideBytes, ptrdiff_t xStrideBytes, ptrdiff_t yStrideBytes)

virtual **~PackedImageDesc**()

ChannelOrdering **getChannelOrder**() const

Get the channel ordering of all the pixels.

virtual *BitDepth* **getBitDepth**() const override

Get the bit-depth.

void ***getData**() const

Get a pointer to the first color channel of the first pixel.

virtual void ***getRData**() const override

Get a pointer to the red channel of the first pixel.

virtual void ***getGData**() const override

Get a pointer to the green channel of the first pixel.

virtual void ***getBData**() const override

Get a pointer to the blue channel of the first pixel.

virtual void ***getAData**() const override

Get a pointer to the alpha channel of the first pixel or null as alpha channel is optional.

virtual long **getWidth**() const override

Get the width to process (where x position starts at 0 and ends at width-1).

virtual long **getHeight**() const override

Get the height to process (where y position starts at 0 and ends at height-1).

long **getNumChannels**() const

ptrdiff_t **getChanStrideBytes**() const

virtual ptrdiff_t **getXStrideBytes**() const override

Get the step in bytes to find the same color channel of the next pixel.

virtual ptrdiff_t **getYStrideBytes**() const override

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

virtual bool **isRGBAPacked**() const override

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

virtual bool **isFloat**() const override

Is the image buffer 32-bit float?

PlanarImageDesc

Python

class PyOpenColorIO.**PlanarImageDesc**

All the constructors expect pointers to the specified image planes (i.e. rrrr gggg bbbb) starting at the first color channel of the first pixel to process (which need not be the first pixel of the image). Pass NULL for aData if no alpha exists (r/g/bData must not be NULL).

Note: The methods assume the CPUProcessor bit-depth type for the R/G/B/A data pointers.

PlanarImageDesc(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.PlanarImageDesc, rData: typing_extensions.Buffer, gData: typing_extensions.Buffer, bData: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.PlanarImageDesc, rData: typing_extensions.Buffer, gData: typing_extensions.Buffer, bData: typing_extensions.Buffer, aData: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt) -> None

Note that although PlanarImageDesc is powerful enough to also describe all PackedImageDesc scenarios, it is recommended to use a PackedImageDesc where possible since that allows for additional optimizations.

3. __init__(self: PyOpenColorIO.PyOpenColorIO.PlanarImageDesc, rData: typing_extensions.Buffer, gData: typing_extensions.Buffer, bData: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, bitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, xStrideBytes: typing.SupportsInt, yStrideBytes: typing.SupportsInt) -> None

4. __init__(self: PyOpenColorIO.PyOpenColorIO.PlanarImageDesc, rData: typing_extensions.Buffer, gData: typing_extensions.Buffer, bData: typing_extensions.Buffer, aData: typing_extensions.Buffer, width: typing.SupportsInt, height: typing.SupportsInt, bitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, xStrideBytes: typing.SupportsInt, yStrideBytes: typing.SupportsInt) -> None

getAData() → numpy.ndarray

Get a pointer to the alpha channel of the first pixel or null as alpha channel is optional.

getBData() → numpy.ndarray

Get a pointer to the blue channel of the first pixel.

getBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Get the bit-depth.

getGData() → numpy.ndarray

Get a pointer to the green channel of the first pixel.

getHeight() → int

Get the height to process (where y position starts at 0 and ends at height-1).

getRData() → numpy.ndarray

Get a pointer to the red channel of the first pixel.

getWidth() → int

Get the width to process (where x position starts at 0 and ends at width-1).

getXStrideBytes() → int

Get the step in bytes to find the same color channel of the next pixel.

getYStrideBytes() → int

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

isFloat() → bool

Is the image buffer 32-bit float?

isRGBAPacked() → bool

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

C++

class **PlanarImageDesc** : public *ImageDesc*

All the constructors expect pointers to the specified image planes (i.e. rrrr gggg bbbb) starting at the first color channel of the first pixel to process (which need not be the first pixel of the image). Pass NULL for aData if no alpha exists (r/g/bData must not be NULL).

Note: The methods assume the CPUProcessor bit-depth type for the R/G/B/A data pointers.

Public Functions

PlanarImageDesc(void *rData, void *gData, void *bData, void *aData, long width, long height)

PlanarImageDesc(void *rData, void *gData, void *bData, void *aData, long width, long height, *BitDepth* bitDepth, ptrdiff_t xStrideBytes, ptrdiff_t yStrideBytes)

Note that although PlanarImageDesc is powerful enough to also describe all PackedImageDesc scenarios, it is recommended to use a PackedImageDesc where possible since that allows for additional optimizations.

virtual **~PlanarImageDesc**()

virtual void ***getRData**() const override

Get a pointer to the red channel of the first pixel.

virtual void ***getGData**() const override

Get a pointer to the green channel of the first pixel.

virtual void ***getBData**() const override

Get a pointer to the blue channel of the first pixel.

virtual void ***getAData**() const override

Get a pointer to the alpha channel of the first pixel or null as alpha channel is optional.

virtual *BitDepth* **getBitDepth**() const override

Get the bit-depth.

virtual long **getWidth**() const override

Get the width to process (where x position starts at 0 and ends at width-1).

virtual long **getHeight**() const override

Get the height to process (where y position starts at 0 and ends at height-1).

virtual ptrdiff_t **getXStrideBytes**() const override

Get the step in bytes to find the same color channel of the next pixel.

virtual ptrdiff_t **getYStrideBytes**() const override

Get the step in bytes to find the same color channel of the pixel at the same position in the next line.

virtual bool **isRGBAPacked**() const override

Is the image buffer in packed mode with the 4 color channels? ("Packed" here means that XStrideBytes is 4x the bytes per channel, so it is more specific than simply any PackedImageDesc.)

virtual bool **isFloat**() const override

Is the image buffer 32-bit float?

5.7.20 Look

Python

class PyOpenColorIO.**Look**

The *:ref:`Look`* is an 'artistic' image modification, in a specified image state. The processSpace defines the ColorSpace the image is required to be in, for the math to apply correctly.

Look(*args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.Look) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.Look, name: str = '', processSpace: str = '', transform: PyOpenColorIO.PyOpenColorIO.Transform = None, inverseTransform: PyOpenColorIO.PyOpenColorIO.Transform = None, description: str = '') -> None

getDescription() → str

getInterchangeAttribute(*attrName: str*) → str

Get/Set the interchange attributes.

Currently the only supported attribute name is "amf_transform_ids". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

getInterchangeAttributes() → dict[str, str]

getInverseTransform() → *PyOpenColorIO.PyOpenColorIO.Transform*

getName() → str

getProcessSpace() → str

getTransform() → *PyOpenColorIO.PyOpenColorIO.Transform*

setDescription(*description: str*) → None

setInterchangeAttribute(*attrName: str*, *attrValue: str*) → None

setInverseTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform) → None

Setting a transform to a non-null call makes it allowed.

setName(*name: str*) → None

setProcessSpace(*processSpace: str*) → None

setTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform) → None

Setting a transform to a non-null call makes it allowed.

C++

class **Look**

The *Look* is an 'artistic' image modification, in a specified image state. The processSpace defines the ColorSpace the image is required to be in, for the math to apply correctly.

Public Functions

LookRcPtr **createEditableCopy**() const

const char ***getName**() const

void **setName**(const char *name)

const char ***getProcessSpace**() const

void **setProcessSpace**(const char *processSpace)

ConstTransformRcPtr **getTransform**() const

void **setTransform**(const *ConstTransformRcPtr* &transform)

Setting a transform to a non-null call makes it allowed.

ConstTransformRcPtr **getInverseTransform**() const

void **setInverseTransform**(const *ConstTransformRcPtr* &transform)

Setting a transform to a non-null call makes it allowed.

const char ***getDescription**() const

void **setDescription**(const char *description)

const char ***getInterchangeAttribute**(const char *attrName) const

Get/Set the interchange attributes.

Currently the only supported attribute name is "amf_transform_ids". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

void **setInterchangeAttribute**(const char *attrName, const char *value)

std::map<std::string, std::string> **getInterchangeAttributes**() const noexcept

Look(const *Look*&) = delete

Look &**operator=**(const *Look*&) = delete

~Look()

Do not use (needed only for pybind11).

Public Static Functions

static *LookRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *Look*&)

typedef std::shared_ptr<const *Look*> OpenColorIO_v2_5::**ConstLookRcPtr**

typedef std::shared_ptr<*Look*> OpenColorIO_v2_5::**LookRcPtr**

5.7.21 Named Transform

Python

class PyOpenColorIO.**NamedTransform**

NamedTransform.

A NamedTransform provides a way for config authors to include a set of color transforms that are independent of the color space being processed. For example a "utility curve" transform where there is no need to convert to or from a reference space.

static GetTransform(*transform:* PyOpenColorIO.PyOpenColorIO.NamedTransform, *direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → *PyOpenColorIO.PyOpenColorIO.Transform*

Will create the transform from the inverse direction if the transform for requested direction is missing.

class NamedTransformAliasIterator

NamedTransformAliasIterator(**args*, ***kwargs*)

NamedTransformAliasIterator(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.NamedTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.NamedTransform, name: str = '', aliases: collections.abc.Sequence[str] = [], family: str = '', description: str = '', forwardTransform: PyOpenColorIO.PyOpenColorIO.Transform = None, inverseTransform: PyOpenColorIO.PyOpenColorIO.Transform = None, categories: collections.abc.Sequence[str] = []) -> None

addAlias(*alias: str*) → None

Nothing is done if alias is NULL or empty, if it is already there, or if it is already the named transform name.

addCategory(*category: str*) → None

clearAliases() → None

clearCategories() → None

getAliases() → *PyOpenColorIO.PyOpenColorIO.NamedTransform.NamedTransformAliasIterator*

getCategories() → *PyOpenColorIO.PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator*

getDescription() → str

getEncoding() → str

A NamedTransform is not a color space and does not have an encoding in the same sense. However, it may be useful to associate a color space encoding that the transform is intended to be used with, for organizational purposes.

getFamily() → str

getName() → str

getTransform(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → *PyOpenColorIO.PyOpenColorIO.Transform*

hasAlias(*alias: str*) → bool

Return true if alias exists.

hasCategory(*category: str*) → bool

removeAlias(*alias: str*) → None

Does nothing if alias is not present.

removeCategory(*category: str*) → None

setDescription(*description: str*) → None

setEncoding(*encoding: str*) → None

setFamily(*family: str*) → None

setName(*name: str*) → None

setTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform, *direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

class PyOpenColorIO.NamedTransform.**NamedTransformCategoryIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator*

len(*self*) → int

next(*self*) → str

C++

class **NamedTransform**

NamedTransform.

A NamedTransform provides a way for config authors to include a set of color transforms that are independent of the color space being processed. For example a "utility

curve" transform where there is no need to convert to or from a reference space.

Public Functions

virtual *NamedTransformRcPtr* **createEditableCopy**() const = 0

virtual const char ***getName**() const noexcept = 0

virtual void **setName**(const char *name) noexcept = 0

virtual size_t **getNumAliases**() const noexcept = 0

Aliases can be used instead of the name. They must be unique within the config.

virtual const char ***getAlias**(size_t idx) const noexcept = 0

Return empty string if idx is out of range.

virtual bool **hasAlias**(const char *alias) const noexcept = 0

Return true if alias exists.

virtual void **addAlias**(const char *alias) noexcept = 0

Nothing is done if alias is NULL or empty, if it is already there, or if it is already the named transform name.

virtual void **removeAlias**(const char *alias) noexcept = 0

Does nothing if alias is not present.

virtual void **clearAliases**() noexcept = 0

virtual const char ***getFamily**() const noexcept = 0

See also:

ColorSpace::getFamily

virtual void **setFamily**(const char *family) noexcept = 0

See also:

ColorSpace::setFamily

virtual const char ***getDescription**() const noexcept = 0

virtual void **setDescription**(const char *description) noexcept = 0

virtual bool **hasCategory**(const char *category) const noexcept = 0

See also:

ColorSpace::hasCategory

virtual void **addCategory**(const char *category) noexcept = 0

See also:

ColorSpace::addCategory

virtual void **removeCategory**(const char *category) noexcept = 0

See also:

ColorSpace::removeCategory

virtual int **getNumCategories**() const noexcept = 0

See also:

ColorSpace::getNumCategories

virtual const char ***getCategory**(int index) const noexcept = 0

See also:

ColorSpace::getCategory

virtual void **clearCategories**() noexcept = 0

See also:

ColorSpace::clearCategories

virtual const char ***getEncoding**() const noexcept = 0

A NamedTransform is not a color space and does not have an encoding in the same sense. However, it may be useful to associate a color space encoding that the transform is intended to be used with, for organizational purposes.

virtual void **setEncoding**(const char *encoding) noexcept = 0

virtual *ConstTransformRcPtr* **getTransform**(*TransformDirection* dir) const = 0

virtual void **setTransform**(const *ConstTransformRcPtr* &transform, *TransformDirection* dir) = 0

NamedTransform(const *NamedTransform*&) = delete

NamedTransform &**operator=**(const *NamedTransform*&) = delete

virtual **~NamedTransform**() = default

Public Static Functions

static *NamedTransformRcPtr* **Create**()

static *ConstTransformRcPtr* **GetTransform**(const *ConstNamedTransformRcPtr* &nt, *TransformDirection* dir)

Will create the transform from the inverse direction if the transform for requested direction is missing.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *NamedTransform*&)

typedef std::shared_ptr<const *NamedTransform*> OpenColorIO_v2_5::**ConstNamedTransformRcPtr**

typedef std::shared_ptr<*NamedTransform*> OpenColorIO_v2_5::**NamedTransformRcPtr**

5.7.22 Processors

Processor

Python

class PyOpenColorIO.**Processor**

The *:ref: `Processor `* represents a specific color transformation which is the result of Config::getProcessor.

Processor(**args*, ***kwargs*)

createGroupTransform() → *PyOpenColorIO.PyOpenColorIO.GroupTransform*

Return a GroupTransform that contains a copy of the transforms that comprise the processor. (Changes to it will not modify the original processor.) Note that the GroupTransform::write method may be used to serialize a Processor. Serializing to CTF format is a useful technique for debugging Processor contents.

getCacheID() → str

Returns a hash string generated by hashing the cachedIDs of the (unoptimized) list of ops contained in the Processor. (This forms part of the key used by the config's processor cache.)

getDefaultCPUProcessor() → *PyOpenColorIO.PyOpenColorIO.CPUProcessor*

Get an optimized CPUProcessor instance.

Note: This may provide higher fidelity than anticipated due to internal optimizations. For example, if the inputColorSpace and the outputColorSpace are members of the same equalitygroup, no conversion will be applied, even though strictly speaking quantization should be added.

Note: The typical use case to apply color processing to an image is:

```
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

OCIO::ConstProcessorRcPtr processor
    = config->getProcessor(colorSpace1, colorSpace2);

OCIO::ConstCPUProcessorRcPtr cpuProcessor
    = processor->getDefaultCPUProcessor();

OCIO::PackedImageDesc img(imgDataPtr, imgWidth, imgHeight, imgChannels);
cpuProcessor->apply(img);
```

getDefaultGPUProcessor() → *PyOpenColorIO.PyOpenColorIO.GPUProcessor*

Get an optimized GPUProcessor instance.

getDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

The returned pointer may be used to set the default value of any dynamic properties of the requested type. Throws if the requested property is not found. Note that if the processor contains several ops that support the requested property, only one can be dynamic and only this one will be controlled.

Note: The dynamic properties are a convenient way to change on-the-fly values without generating again and again a CPU or GPU processor instance. Color transformations can contain dynamic properties from a ExposureContrastTransform for example. So, Processor, CPUProcessor and GpuShaderCreator all have

ways to manage dynamic properties. However, the transform dynamic properties are decoupled between the types of processor instances so that the same Processor can generate several independent CPU and/or GPU processor instances i.e. changing the value of the exposure dynamic property from a CPU processor instance does not affect the corresponding GPU processor instance. Processor creation will log a warning if there are more than one property of a given type. There may be more than one property of a given type, but only one will respond to parameter updates, the others will use their original parameter values.

---

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

Get a FormatMetadata containing the top level metadata for the processor. For a processor from a CLF file, this corresponds to the ProcessList metadata.

getOptimizedCPUProcessor(**args*, ***kwargs*)

Overloaded function.

1. getOptimizedCPUProcessor(self: PyOpenColorIO.PyOpenColorIO.Processor, oFlags: PyOpenColorIO.PyOpenColorIO.OptimizationFlags) -> PyOpenColorIO.PyOpenColorIO.CPUProcessor

2. getOptimizedCPUProcessor(self: PyOpenColorIO.PyOpenColorIO.Processor, inBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, outBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, oFlags: PyOpenColorIO.PyOpenColorIO.OptimizationFlags) -> PyOpenColorIO.PyOpenColorIO.CPUProcessor

getOptimizedGPUProcessor(*oFlags:* PyOpenColorIO.PyOpenColorIO.OptimizationFlags) → *PyOpenColorIO.PyOpenColorIO.GPUProcessor*

getOptimizedProcessor(**args*, ***kwargs*)

Overloaded function.

1. getOptimizedProcessor(self: PyOpenColorIO.PyOpenColorIO.Processor, oFlags: PyOpenColorIO.PyOpenColorIO.OptimizationFlags) -> PyOpenColorIO.PyOpenColorIO.Processor

Run the optimizer on a Processor to create a new Processor. It is usually not necessary to call this since getting a CPUProcessor or GPUProcessor will also optimize. However if you need both, calling this method first makes getting a CPU and GPU Processor faster since the optimization is effectively only done once.

2. getOptimizedProcessor(self: PyOpenColorIO.PyOpenColorIO.Processor, inBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, outBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth, oFlags: PyOpenColorIO.PyOpenColorIO.OptimizationFlags) -> PyOpenColorIO.PyOpenColorIO.Processor

Run the optimizer on a Processor to create a new Processor. It is usually not necessary to call this since getting a CPUProcessor or GPUProcessor will also optimize. However if you need both, calling this method first makes getting a CPU and GPU Processor faster since the optimization is effectively only done once.

getProcessorMetadata() → *PyOpenColorIO.PyOpenColorIO.ProcessorMetadata*

The ProcessorMetadata contains technical information such as the number of files and looks used in the processor.

getTransformFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.Processor.TransformFormatMetadataIterator*

hasChannelCrosstalk() → bool

True if the image transformation is non-separable. For example, if a change in red may also cause a change in green or blue.

hasDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → bool

True if at least one dynamic property of that type exists.

isDynamic() → bool

True if at least one dynamic property of any type exists and is dynamic.

isNoOp() → bool

class PyOpenColorIO.Processor.**TransformFormatMetadataIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.Processor.TransformFormatMetadataIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

C++

class **Processor**

The *Processor* represents a specific color transformation which is the result of Config::getProcessor.

Public Functions

bool **isNoOp**() const

bool **hasChannelCrosstalk**() const

True if the image transformation is non-separable. For example, if a change in red may also cause a change in green or blue.

const char ***getCacheID**() const

Returns a hash string generated by hashing the cachedIDs of the (unoptimized) list of ops contained in the Processor. (This forms part of the key used by the config's processor cache.)

ConstProcessorMetadataRcPtr **getProcessorMetadata**() const

The ProcessorMetadata contains technical information such as the number of files and looks used in the processor.

const *FormatMetadata* &**getFormatMetadata**() const

Get a FormatMetadata containing the top level metadata for the processor. For a processor from a CLF file, this corresponds to the ProcessList metadata.

int **getNumTransforms**() const

Get the number of transforms that comprise the processor. Each transform has a (potentially empty) FormatMetadata.

const *FormatMetadata* &**getTransformFormatMetadata**(int index) const

Get a FormatMetadata containing the metadata for a transform within the processor. For a processor from a CLF file, this corresponds to the metadata associated with an individual process node.

GroupTransformRcPtr **createGroupTransform**() const

Return a GroupTransform that contains a copy of the transforms that comprise the processor. (Changes to it will not modify the original processor.) Note that the GroupTransform::write method may be used to serialize a Processor. Serializing to CTF format is a useful technique for debugging Processor contents.

DynamicPropertyRcPtr **getDynamicProperty**(*DynamicPropertyType* type) const

The returned pointer may be used to set the default value of any dynamic properties of the requested type. Throws if the requested property is not found. Note that if the processor contains several ops that support the requested property, only one can be dynamic and only this one will be controlled.

Note: The dynamic properties are a convenient way to change on-the-fly values without generating again and again a CPU or GPU processor instance. Color transformations can contain dynamic properties from a ExposureContrastTransform for example. So, Processor, CPUProcessor and GpuShaderCreator all have ways to manage dynamic properties. However, the transform dynamic properties are decoupled between the types of processor instances so that the same Processor can generate several independent CPU and/or GPU processor instances i.e. changing the value of the exposure dynamic property from a CPU processor instance does not affect the corresponding GPU processor instance. Processor creation will log a warning if there are more than one property of a given type. There may be more than one property of a given type, but only one will respond to parameter updates, the others will use their original parameter values.

---

bool **hasDynamicProperty**(*DynamicPropertyType* type) const noexcept

True if at least one dynamic property of that type exists.

bool **isDynamic**() const noexcept

True if at least one dynamic property of any type exists and is dynamic.

ConstProcessorRcPtr **getOptimizedProcessor**(*OptimizationFlags* oFlags) const

Run the optimizer on a Processor to create a new Processor. It is usually not necessary to call this since getting a CPUProcessor or GPUProcessor will also optimize. However if you need both, calling this method first makes getting a CPU and GPU Processor faster since the optimization is effectively only done once.

ConstProcessorRcPtr **getOptimizedProcessor**(*BitDepth* inBD, *BitDepth* outBD, *OptimizationFlags* oFlags) const

Create a Processor that is optimized for a specific in and out bit-depth (as CPUProcessor would do). This method is provided primarily for diagnostic purposes.

ConstGPUProcessorRcPtr **getDefaultGPUProcessor**() const

Get an optimized GPUProcessor instance.

ConstGPUProcessorRcPtr **getOptimizedGPUProcessor**(*OptimizationFlags* oFlags) const

ConstGPUProcessorRcPtr **getOptimizedLegacyGPUProcessor**(*OptimizationFlags* oFlags, unsigned edgelen) const

Get an optimized GPUProcessor instance that will emulate the OCIO v1 GPU path. This approach bakes some of the ops into a single Lut3D and so is less accurate than the current GPU processing methods.

ConstCPUProcessorRcPtr **getDefaultCPUProcessor**() const

Get an optimized CPUProcessor instance.

```
OCIO::ConstConfigRcPtr config = OCIO::GetCurrentConfig();

OCIO::ConstProcessorRcPtr processor
    = config->getProcessor(colorSpace1, colorSpace2);

OCIO::ConstCPUProcessorRcPtr cpuProcessor
    = processor->getDefaultCPUProcessor();
```

(continues on next page)

(continued from previous page)

```
OCIO::PackedImageDesc img(imgDataPtr, imgWidth, imgHeight, imgChannels);
cpuProcessor->apply(img);
```

Note: This may provide higher fidelity than anticipated due to internal optimizations. For example, if the inputColorSpace and the outputColorSpace are members of the same equalitygroup, no conversion will be applied, even though strictly speaking quantization should be added.

Note: The typical use case to apply color processing to an image is:

ConstCPUProcessorRcPtr **getOptimizedCPUProcessor**(*OptimizationFlags* oFlags) const

ConstCPUProcessorRcPtr **getOptimizedCPUProcessor**(*BitDepth* inBitDepth, *BitDepth* outBitDepth, *OptimizationFlags* oFlags) const

Processor(const *Processor*&) = delete

Processor &**operator=**(const *Processor*&) = delete

~Processor()

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *Processor*> OpenColorIO_v2_5::**ConstProcessorRcPtr**

typedef std::shared_ptr<*Processor*> OpenColorIO_v2_5::**ProcessorRcPtr**

CPUProcessor

Python

class PyOpenColorIO.**CPUProcessor**

CPUProcessor(**args*, ***kwargs*)

apply(**args*, ***kwargs*)

Overloaded function.

1. apply(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, imgDesc: PyOpenColorIO.PyOpenColorIO.ImageDesc) -> None

Apply to an image with any kind of channel ordering while respecting the input and output bit-depths. Image values are modified in place.

Note: The GIL is released during processing, freeing up Python to execute other threads concurrently.

Note: For large images, `applyRGB` or `applyRGBA` are preferred for processing a NumPy array. The Python `ImageDesc` implementation requires copying all values (once) in order to own the underlying pointer. The dedicated packed `apply*` methods utilize `ImageDesc` on the C++ side so avoid the copy.

2. apply(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, srcImgDesc: PyOpenColorIO.PyOpenColorIO.ImageDesc, dstImgDesc: PyOpenColorIO.PyOpenColorIO.ImageDesc) -> None

Apply to an image with any kind of channel ordering while respecting the input and output bit-depths. Modified srcImgDesc image values are written to the dstImgDesc image, leaving srcImgDesc unchanged.

Note: The GIL is released during processing, freeing up Python to execute other threads concurrently.

Note: For large images, `applyRGB` or `applyRGBA` are preferred for processing a NumPy array. The Python `ImageDesc` implementation requires copying all values (once) in order to own the underlying pointer. The dedicated packed `apply*` methods utilize `ImageDesc` on the C++ side so avoid the copy.

---

applyRGB(**args*, ***kwargs*)

Overloaded function.

1. applyRGB(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, data: typing_extensions.Buffer) -> None

Apply to a packed RGB array adhering to the Python buffer protocol. This will typically be a NumPy array. Input and output bit-depths are respected but must match. Any array size or shape is supported as long as the flattened array size is divisible by 3. Array values are modified in place.

Note: This differs from the C++ implementation which only applies to a single pixel. This method uses a `PackedImageDesc` under the hood to apply to an entire image at once. The GIL is released during processing, freeing up Python to execute other threads concurrently.

---

2. applyRGB(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, data: collections.abc.Sequence[typing.SupportsFloat]) -> list[float]

Apply to a packed RGB list of float values. Any size is supported as long as the list length is divisible by 3. A new list with processed float values is returned, leaving the input list unchanged.

Note: This differs from the C++ implementation which only applies to a single pixel. This method uses a `PackedImageDesc` under the hood to apply to an entire image at once. The GIL is released during processing, freeing up Python to execute other threads concurrently.

Note: For large images, a NumPy array should be preferred over a list. List values are copied on input and output, where an array is modified in place.

---

applyRGBA(**args*, ***kwargs*)

Overloaded function.

1. applyRGBA(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, data: typing_extensions.Buffer) -> None

Apply to a packed RGBA array adhering to the Python buffer protocol. This will typically be a NumPy array. Input and output bit-depths are respected but must match. Any array size or shape is supported as long as the flattened array size is divisible by 4. Array values are modified in place.

Note: This differs from the C++ implementation which only applies to a single pixel. This method uses a `PackedImageDesc` under the hood to apply to an entire image at once. The GIL is released during processing, freeing up Python to execute other threads concurrently.

2. applyRGBA(self: PyOpenColorIO.PyOpenColorIO.CPUProcessor, data: collections.abc.Sequence[typing.SupportsFloat]) -> list[float]

Apply to a packed RGBA list of float values. Any size is supported as long as the list length is divisible by 4. A new list with processed float values is returned, leaving the input list unchanged.

Note: This differs from the C++ implementation which only applies to a single pixel. This method uses a `PackedImageDesc` under the hood to apply to an entire image at once. The GIL is released during processing, freeing up Python to execute other threads concurrently.

Note: For large images, a NumPy array should be preferred over a list. List values are copied on input and output, where an array is modified in place.

getCacheID() → str

getDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

The returned pointer may be used to set the value of any dynamic properties of the requested type. Throws if the requested property is not found. Note that if the processor contains several ops that support the requested property, only one can be dynamic.

Note: The dynamic properties in this object are decoupled from the ones in the Processor it was generated from. For each dynamic property in the Processor, there is one in the CPU processor.

getInputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Bit-depth of the input pixel buffer.

getOutputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Bit-depth of the output pixel buffer.

hasChannelCrosstalk() → bool

hasDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → bool

True if at least one dynamic property of that type exists.

isDynamic() → bool

True if at least one dynamic property of any type exists and is dynamic.

isIdentity() → bool

Equivalent to isNoOp from the underlying Processor, i.e., it ignores in/out bit-depth differences.

isNoOp() → bool

The in and out bit-depths must be equal for isNoOp to be true.

C++

class **CPUProcessor**

Public Functions

bool **isNoOp**() const

The in and out bit-depths must be equal for isNoOp to be true.

bool **isIdentity**() const

Equivalent to isNoOp from the underlying Processor, i.e., it ignores in/out bit-depth differences.

bool **hasChannelCrosstalk**() const

const char ***getCacheID**() const

BitDepth **getInputBitDepth**() const

Bit-depth of the input pixel buffer.

BitDepth **getOutputBitDepth**() const

Bit-depth of the output pixel buffer.

DynamicPropertyRcPtr **getDynamicProperty**(*DynamicPropertyType* type) const

The returned pointer may be used to set the value of any dynamic properties of the requested type. Throws if the requested property is not found. Note that if the processor contains several ops that support the requested property, only one can be dynamic.

Note: The dynamic properties in this object are decoupled from the ones in the Processor it was generated from. For each dynamic property in the Processor, there is one in the CPU processor.

---

bool **hasDynamicProperty**(*DynamicPropertyType* type) const noexcept

True if at least one dynamic property of that type exists.

bool **isDynamic**() const noexcept

True if at least one dynamic property of any type exists and is dynamic.

void **apply**(const *ImageDesc* &imgDesc) const

Apply to an image with any kind of channel ordering while respecting the input and output bit-depths.

void **apply**(const *ImageDesc* &srcImgDesc, *ImageDesc* &dstImgDesc) const

void **applyRGB**(float *pixel) const

Apply to a single pixel respecting that the input and output bit-depths be 32-bit float and the image buffer be packed RGB/RGBA.

Note: This is not as efficient as applying to an entire image at once. If you are processing multiple pixels, and have the flexibility, use the above function instead.

---

void **applyRGBA**(float *pixel) const

CPUProcessor(const *CPUProcessor*&) = delete

CPUProcessor &**operator=**(const *CPUProcessor*&) = delete

~CPUProcessor()

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *CPUProcessor*> OpenColorIO_v2_5::**ConstCPUProcessorRcPtr**

typedef std::shared_ptr<*CPUProcessor*> OpenColorIO_v2_5::**CPUProcessorRcPtr**

GPUProcessor

Python

class PyOpenColorIO.**GPUProcessor**

GPUProcessor(**args*, ***kwargs*)

extractGpuShaderInfo(*shaderDesc:* PyOpenColorIO.PyOpenColorIO.GpuShaderDesc) → None

Extract & Store the shader information to implement the color processing.

getCacheID() → str

hasChannelCrosstalk() → bool

isNoOp() → bool

C++

class **GPUProcessor**

Public Functions

bool **isNoOp**() const

bool **hasChannelCrosstalk**() const

const char ***getCacheID**() const

void **extractGpuShaderInfo**(*GpuShaderDescRcPtr* &shaderDesc) const

Extract & Store the shader information to implement the color processing.

void **extractGpuShaderInfo**(*GpuShaderCreatorRcPtr* &shaderCreator) const

Extract the shader information using a custom GpuShaderCreator class.

GPUProcessor(const *GPUProcessor*&) = delete

GPUProcessor &**operator=**(const *GPUProcessor*&) = delete

~GPUProcessor()

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *GPUProcessor*> OpenColorIO_v2_5::**ConstGPUProcessorRcPtr**

typedef std::shared_ptr<*GPUProcessor*> OpenColorIO_v2_5::**GPUProcessorRcPtr**

ProcessorMetadata

Python

class PyOpenColorIO.**ProcessorMetadata**

This class contains meta information about the process that generated this processor. The results of these functions do not impact the pixel processing.

ProcessorMetadata → None

addFile(*fileName: str*) → None

addLook(*look: str*) → None

getFiles() → *PyOpenColorIO.PyOpenColorIO.ProcessorMetadata.FileIterator*

getLooks() → *PyOpenColorIO.PyOpenColorIO.ProcessorMetadata.LookIterator*

class PyOpenColorIO.ProcessorMetadata.**FileIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ProcessorMetadata.FileIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.ProcessorMetadata.**LookIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ProcessorMetadata.LookIterator*

len(*self*) → int

next(*self*) → str

C++

class **ProcessorMetadata**

This class contains meta information about the process that generated this processor. The results of these functions do not impact the pixel processing.

Public Functions

int **getNumFiles**() const

const char ***getFile**(int index) const

int **getNumLooks**() const

const char ***getLook**(int index) const

void **addFile**(const char *fname)

void **addLook**(const char *look)

ProcessorMetadata(const *ProcessorMetadata*&) = delete

ProcessorMetadata &**operator=**(const *ProcessorMetadata*&) = delete

~ProcessorMetadata()

Do not use (needed only for pybind11).

Public Static Functions

static *ProcessorMetadataRcPtr* **Create**()

typedef std::shared_ptr<const *ProcessorMetadata*> OpenColorIO_v2_5::**ConstProcessorMetadataRcPtr**

typedef std::shared_ptr<*ProcessorMetadata*> OpenColorIO_v2_5::**ProcessorMetadataRcPtr**

5.7.23 Rules

FileRules

Python

class PyOpenColorIO.**FileRules**

The File Rules are a set of filepath to color space mappings that are evaluated from first to last. The first rule to match is what determines which color space is returned. There are four types of rules available. Each rule type has a name key that may be used by applications to refer to that rule. Name values must be unique i.e. using a case insensitive comparison. The other keys depend on the rule type:

- *Basic Rule*: This is the basic rule type that uses Unix glob style pattern matching and is thus very easy to use. It contains the keys:
- name: Name of the rule
- colorspace: Color space name to be returned.
- pattern: Glob pattern to be used for the main part of the name/path.
- extension: Glob pattern to be used for the file extension. Note that if glob tokens are not used, the extension will be used in a non-case-sensitive way by default.
- *Regex Rule*: This is similar to the basic rule but allows additional capabilities for power-users. It contains the keys:
- name: Name of the rule
- colorspace: Color space name to be returned.
- regex: Regular expression to be evaluated.
- *OCIO v1 style Rule*: This rule allows the use of the OCIO v1 style, where the string is searched for color space names from the config. This rule may occur 0 or 1 times in the list. The position in the list prioritizes it with respect to the other rules. StrictParsing is not used. If no color space is found in the path, the rule will not match and the next rule will be considered. see insertPathSearchRule. It has the key:
- name: Must be "ColorSpaceNamePathSearch".
- *Default Rule*: The file_rules must always end with this rule. If no prior rules match, this rule specifies the color space applications will use. see setDefaultRuleColorSpace. It has the keys:

- name: must be “Default”.
- colorspace : Color space name to be returned.

Custom string keys and associated string values may be used to convey app or workflow-specific information, e.g. whether the color space should be left as is or converted into a working space.

Getters and setters are using the rule position, they will throw if the position is not valid. If the rule at the specified position does not implement the requested property getter will return NULL and setter will throw.

When loading a v1 config, a set of FileRules are created with ColorSpaceNamePathSearch followed by the Default rule pointing to the default role. This allows getColorSpaceFromFilepath to emulate OCIO v1 code that used parseColorSpaceFromString with strictparsing set to false.

FileRules → None

Creates FileRules for a Config. File rules will contain the default rule using the default role. The default rule cannot be removed.

decreaseRulePriority(*ruleIndex: SupportsInt*) → None

Move a rule closer to the end of the list by one position.

getColorSpace(*ruleIndex: SupportsInt*) → str

Set the rule’s color space (may also be a role).

getCustomKeyName(*ruleIndex: SupportsInt*, *key: SupportsInt*) → str

Get name of key.

getCustomKeyValue(*ruleIndex: SupportsInt*, *key: SupportsInt*) → str

Get value for the key.

getExtension(*ruleIndex: SupportsInt*) → str

Setting extension will erase regex.

getIndexForRule(*ruleName: str*) → int

Get the index from the rule name. Throws if the rule is not found.

getName(*ruleIndex: SupportsInt*) → str

Get name of the rule.

getNumCustomKeys(*ruleIndex: SupportsInt*) → int

Get number of key/value pairs.

getNumEntries() → int

Does include default rule. Result will be at least 1.

getPattern(*ruleIndex: SupportsInt*) → str

Setting pattern will erase regex.

getRegex(*ruleIndex: SupportsInt*) → str

Setting a regex will erase pattern & extension.

increaseRulePriority(*ruleIndex: SupportsInt*) → None

Move a rule closer to the start of the list by one position.

insertPathSearchRule(*ruleIndex: SupportsInt*) → None

Helper function to insert a rule.

Uses Config:parseColorSpaceFromString to search the path for any of the color spaces named in the config (as per OCIO v1).

insertRule(**args*, ***kwargs*)

Overloaded function.

1. insertRule(self: PyOpenColorIO.PyOpenColorIO.FileRules, ruleIndex: typing.SupportsInt, name: str, colorSpace: str, pattern: str, extension: str) -> None

Insert a rule at a given ruleIndex.

Rule currently at ruleIndex will be pushed to index: ruleIndex + 1. Name must be unique. - "Default" is a reserved name for the default rule. The default rule is automatically added and can't be removed. (see FileRules::setDefaultRuleColorSpace). - "ColorSpaceNamePathSearch" is also a reserved name (see FileRules::insertPathSearchRule).

Will throw if pattern, extension or regex is a null or empty string.

Will throw if ruleIndex is not less than FileRules::getNumEntries .

2. insertRule(self: PyOpenColorIO.PyOpenColorIO.FileRules, ruleIndex: typing.SupportsInt, name: str, colorSpace: str, regex: str) -> None

isDefault() → bool

Check if there is only the default rule using default role and no custom key. This is the default FileRules state when creating a new config.

removeRule(*ruleIndex: SupportsInt*) → None

Note: Default rule can't be removed. Will throw if ruleIndex + 1 is not less than FileRules::getNumEntries .

---

setColorSpace(*ruleIndex: SupportsInt*, *colorSpace: str*) → None

setCustomKey(*ruleIndex: SupportsInt*, *key: str*, *value: str*) → None

Adds a key/value or replace value if key exists. Setting a NULL or an empty value will erase the key.

setDefaultRuleColorSpace(*colorSpace: str*) → None

Helper function to set the color space for the default rule.

setExtension(*ruleIndex: SupportsInt*, *extension: str*) → None

setPattern(*ruleIndex: SupportsInt*, *pattern: str*) → None

setRegex(*ruleIndex: SupportsInt*, *regex: str*) → None

C++

class **FileRules**

The File Rules are a set of filepath to color space mappings that are evaluated from first to last. The first rule to match is what determines which color space is returned. There are four types of rules available. Each rule type has a name key that may be used by applications to refer to that rule. Name values must be unique i.e. using a case insensitive comparison. The other keys depend on the rule type:

- *Basic Rule*: This is the basic rule type that uses Unix glob style pattern matching and is thus very easy to use. It contains the keys:
  - name: Name of the rule
  - colorspace: Color space name to be returned.

  - pattern: Glob pattern to be used for the main part of the name/path.
  - extension: Glob pattern to be used for the file extension. Note that if glob tokens are not used, the extension will be used in a non-case-sensitive way by default.

- *Regex Rule*: This is similar to the basic rule but allows additional capabilities for power-users. It contains the keys:
  - name: Name of the rule
  - colorspace: Color space name to be returned.
  - regex: Regular expression to be evaluated.
- *OCIO v1 style Rule*: This rule allows the use of the OCIO v1 style, where the string is searched for color space names from the config. This rule may occur 0 or 1 times in the list. The position in the list prioritizes it with respect to the other rules. StrictParsing is not used. If no color space is found in the path, the rule will not match and the next rule will be considered. see insertPathSearchRule. It has the key:
  - name: Must be "ColorSpaceNamePathSearch".
- *Default Rule*: The file_rules must always end with this rule. If no prior rules match, this rule specifies the color space applications will use. see setDefaultRuleColorSpace. It has the keys:
  - name: must be "Default".
  - colorspace : Color space name to be returned.

Custom string keys and associated string values may be used to convey app or workflow-specific information, e.g. whether the color space should be left as is or converted into a working space.

Getters and setters are using the rule position, they will throw if the position is not valid. If the rule at the specified position does not implement the requested property getter will return NULL and setter will throw.

When loading a v1 config, a set of FileRules are created with ColorSpaceNamePathSearch followed by the Default rule pointing to the default role. This allows getColorSpaceFromFilepath to emulate OCIO v1 code that used parseColorSpaceFromString with strictparsing set to false.

Public Functions

FileRulesRcPtr **createEditableCopy**() const

The method clones the content decoupling the two instances.

size_t **getNumEntries**() const noexcept

Does include default rule. Result will be at least 1.

size_t **getIndexForRule**(const char *ruleName) const

Get the index from the rule name. Throws if the rule is not found.

const char ***getName**(size_t ruleIndex) const

Get name of the rule.

const char ***getPattern**(size_t ruleIndex) const

Setting pattern will erase regex.

void **setPattern**(size_t ruleIndex, const char *pattern)

const char ***getExtension**(size_t ruleIndex) const

Setting extension will erase regex.

void **setExtension**(size_t ruleIndex, const char *extension)

const char ***getRegex**(size_t ruleIndex) const

Setting a regex will erase pattern & extension.

void **setRegex**(size_t ruleIndex, const char *regex)

const char ***getColorSpace**(size_t ruleIndex) const

Set the rule's color space (may also be a role).

void **setColorSpace**(size_t ruleIndex, const char *colorSpace)

size_t **getNumCustomKeys**(size_t ruleIndex) const

Get number of key/value pairs.

const char ***getCustomKeyName**(size_t ruleIndex, size_t key) const

Get name of key.

const char ***getCustomKeyValue**(size_t ruleIndex, size_t key) const

Get value for the key.

void **setCustomKey**(size_t ruleIndex, const char *key, const char *value)

Adds a key/value or replace value if key exists. Setting a NULL or an empty value will erase the key.

void **insertRule**(size_t ruleIndex, const char *name, const char *colorSpace, const char *pattern, const char *extension)

Insert a rule at a given ruleIndex.

Rule currently at ruleIndex will be pushed to index: ruleIndex + 1. Name must be unique.

- "Default" is a reserved name for the default rule. The default rule is automatically added and can't be removed. (see FileRules::setDefaultRuleColorSpace).
- "ColorSpaceNamePathSearch" is also a reserved name (see FileRules::insertPathSearchRule).

Will throw if pattern, extension or regex is a null or empty string.

Will throw if ruleIndex is not less than FileRules::getNumEntries .

void **insertRule**(size_t ruleIndex, const char *name, const char *colorSpace, const char *regex)

void **insertPathSearchRule**(size_t ruleIndex)

Helper function to insert a rule.

Uses Config:parseColorSpaceFromString to search the path for any of the color spaces named in the config (as per OCIO v1).

void **setDefaultRuleColorSpace**(const char *colorSpace)

Helper function to set the color space for the default rule.

void **removeRule**(size_t ruleIndex)

Note: Default rule can't be removed. Will throw if ruleIndex + 1 is not less than FileRules::getNumEntries .

---

void **increaseRulePriority**(size_t ruleIndex)

Move a rule closer to the start of the list by one position.

void **decreaseRulePriority**(size_t ruleIndex)

Move a rule closer to the end of the list by one position.

bool **isDefault**() const noexcept

Check if there is only the default rule using default role and no custom key. This is the default FileRules state when creating a new config.

FileRules(const *FileRules*&) = delete

FileRules &**operator=**(const *FileRules*&) = delete

virtual **~FileRules**()

Do not use (needed only for pybind11).

Public Static Functions

static *FileRulesRcPtr* **Create**()

Creates FileRules for a Config. File rules will contain the default rule using the default role. The default rule cannot be removed.

Public Static Attributes

static const char ***DefaultRuleName**

Reserved rule name for the default rule.

static const char ***FilePathSearchRuleName**

Reserved rule name for the file path search rule.

See also:

FileRules::insertPathSearchRule.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *FileRules*&)

typedef std::shared_ptr<const *FileRules*> OpenColorIO_v2_5::**ConstFileRulesRcPtr**

typedef std::shared_ptr<*FileRules*> OpenColorIO_v2_5::**FileRulesRcPtr**

ViewingRules

Python

class PyOpenColorIO.**ViewingRules**

ViewingRules

Viewing Rules allow config authors to filter the list of views an application should offer based on the color space of an image. For example, a config may define a large number of views but not all of them may be appropriate for use with all color spaces. E.g., some views may be intended for use with scene-linear color space encodings and others with video color space encodings.

Each rule has a name key for applications to refer to the rule. Name values must be unique (using case insensitive comparison). Viewing Rules may also have the following keys:

- colorspaces: Either a single colorspace name or a list of names.

- encodings: One or more strings to be found in the colorspace's encoding attribute. Either this attribute or colorspaces must be present, but not both.
- custom : Allows arbitrary key / value string pairs, similar to FileRules.

Getters and setters are using the rule position, they will throw if the position is not valid.

ViewingRules → None

Creates ViewingRules for a Config.

addColorSpace(*ruleIndex: SupportsInt*, *colorSpaceName: str*) → None

Add colorspace name. Will throw if:

- RuleIndex is invalid.
- ViewingRules::getNumEncodings is not 0.

addEncoding(*ruleIndex: SupportsInt*, *encodingName: str*) → None

Add encoding name. Will throw if:

- RuleIndex is invalid.
- ViewingRules::getNumColorSpaces is not 0.

getColorSpaces(*ruleIndex: SupportsInt*) → *PyOpenColorIO.PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator*

getCustomKeyName(*ruleIndex: SupportsInt*, *key: SupportsInt*) → str

Get name of key. Will throw if ruleIndex or keyIndex is invalid.

getCustomKeyValue(*ruleIndex: SupportsInt*, *key: SupportsInt*) → str

Get value for the key. Will throw if ruleIndex or keyIndex is invalid.

getEncodings(*ruleIndex: SupportsInt*) → *PyOpenColorIO.PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator*

getIndexForRule(*ruleName: str*) → int

Get the index from the rule name. Will throw if there is no rule named ruleName.

getName(*ruleIndex: SupportsInt*) → str

Get name of the rule. Will throw if ruleIndex is invalid.

getNumCustomKeys(*ruleIndex: SupportsInt*) → int

Get number of key/value pairs. Will throw if ruleIndex is invalid.

getNumEntries() → int

insertRule(*ruleIndex: SupportsInt*, *name: str*) → None

Insert a rule at a given ruleIndex.

Rule currently at ruleIndex will be pushed to index: ruleIndex + 1. If ruleIndex is ViewingRules::getNumEntries, a new rule will be added at the end. Will throw if: - RuleIndex is invalid (must be less than or equal to ViewingRules::getNumEntries). - RuleName already exists.

removeColorSpace(*ruleIndex: SupportsInt*, *colorSpaceIndex: SupportsInt*) → None

Remove colorspace. Will throw if ruleIndex or colorSpaceIndex is invalid.

removeEncoding(*ruleIndex: SupportsInt*, *encodingIndex: SupportsInt*) → None

Remove encoding. Will throw if ruleIndex or encodingIndex is invalid.

removeRule(*ruleIndex: SupportsInt*) → None

Remove a rule. Throws if ruleIndex is not valid.

setCustomKey(*ruleIndex: SupportsInt*, *key: str*, *value: str*) → None

Adds a key/value or replace value if key exists. Setting a NULL or an empty value will erase the key. Will throw if ruleIndex is invalid.

class PyOpenColorIO.ViewingRules.**ViewingRuleColorSpaceIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.ViewingRules.**ViewingRuleEncodingIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator*

len(*self*) → int

next(*self*) → str

C++

class **ViewingRules**

ViewingRules

Viewing Rules allow config authors to filter the list of views an application should offer based on the color space of an image. For example, a config may define a large number of views but not all of them may be appropriate for use with all color spaces. E.g., some views may be intended for use with scene-linear color space encodings and others with video color space encodings.

Each rule has a name key for applications to refer to the rule. Name values must be unique (using case insensitive comparison). Viewing Rules may also have the following keys:

- colorspaces: Either a single colorspace name or a list of names.
- encodings: One or more strings to be found in the colorspace's encoding attribute. Either this attribute or colorspaces must be present, but not both.
- custom : Allows arbitrary key / value string pairs, similar to FileRules.

Getters and setters are using the rule position, they will throw if the position is not valid.

Public Functions

ViewingRulesRcPtr **createEditableCopy**() const
The method clones the content decoupling the two instances.

size_t **getNumEntries**() const noexcept

size_t **getIndexForRule**(const char *ruleName) const
Get the index from the rule name. Will throw if there is no rule named ruleName.

const char ***getName**(size_t ruleIndex) const
Get name of the rule. Will throw if ruleIndex is invalid.

size_t **getNumColorSpaces**(size_t ruleIndex) const
Get number of colorspaces. Will throw if ruleIndex is invalid.

const char ***getColorSpace**(size_t ruleIndex, size_t colorSpaceIndex) const
Get colorspace name. Will throw if ruleIndex or colorSpaceIndex is invalid.

void **addColorSpace**(size_t ruleIndex, const char *colorSpace)
Add colorspace name. Will throw if:

- RuleIndex is invalid.
- ViewingRules::getNumEncodings is not 0.

void **removeColorSpace**(size_t ruleIndex, size_t colorSpaceIndex)
Remove colorspace. Will throw if ruleIndex or colorSpaceIndex is invalid.

size_t **getNumEncodings**(size_t ruleIndex) const
Get number of encodings. Will throw if ruleIndex is invalid.

const char ***getEncoding**(size_t ruleIndex, size_t encodingIndex) const
Get encoding name. Will throw if ruleIndex or encodingIndex is invalid.

void **addEncoding**(size_t ruleIndex, const char *encoding)
Add encoding name. Will throw if:

- RuleIndex is invalid.
- ViewingRules::getNumColorSpaces is not 0.

void **removeEncoding**(size_t ruleIndex, size_t encodingIndex)
Remove encoding. Will throw if ruleIndex or encodingIndex is invalid.

size_t **getNumCustomKeys**(size_t ruleIndex) const
Get number of key/value pairs. Will throw if ruleIndex is invalid.

const char ***getCustomKeyName**(size_t ruleIndex, size_t keyIndex) const
Get name of key. Will throw if ruleIndex or keyIndex is invalid.

const char ***getCustomKeyValue**(size_t ruleIndex, size_t keyIndex) const
Get value for the key. Will throw if ruleIndex or keyIndex is invalid.

void **setCustomKey**(size_t ruleIndex, const char *key, const char *value)
Adds a key/value or replace value if key exists. Setting a NULL or an empty value will erase the key. Will throw if ruleIndex is invalid.

void **insertRule**(size_t ruleIndex, const char *ruleName)

Insert a rule at a given ruleIndex.

Rule currently at ruleIndex will be pushed to index: ruleIndex + 1. If ruleIndex is ViewingRules::getNumEntries, a new rule will be added at the end. Will throw if:

- RuleIndex is invalid (must be less than or equal to ViewingRules::getNumEntries).
- RuleName already exists.

void **removeRule**(size_t ruleIndex)

Remove a rule. Throws if ruleIndex is not valid.

ViewingRules(const *ViewingRules*&) = delete

ViewingRules &**operator=**(const *ViewingRules*&) = delete

virtual **~ViewingRules**()

Do not use (needed only for pybind11).

Public Static Functions

static *ViewingRulesRcPtr* **Create**()

Creates ViewingRules for a Config.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ViewingRules*&)

typedef std::shared_ptr<const *ViewingRules*> OpenColorIO_v2_5::**ConstViewingRulesRcPtr**

typedef std::shared_ptr<*ViewingRules*> OpenColorIO_v2_5::**ViewingRulesRcPtr**

5.7.24 Shaders

GpuShaderCreator

Python

class PyOpenColorIO.**GpuShaderCreator**

Inherit from the class to fully customize the implementation of a GPU shader program from a color transformation.

When no customizations are needed and the intermediate in-memory step is acceptable then the GpuShaderDesc is a better choice.

Note: To better decouple the DynamicProperties from their GPU implementation, the code provides several addUniform() methods i.e. one per access function types. For example, an ExposureContrastTransform instance owns three DynamicProperties and they are all implemented by a double. When creating the GPU fragment shader program, the addUniform() with GpuShaderCreator::DoubleGetter is called when property is dynamic, up to three times.

An OCIO shader program could contain:

- A declaration part e.g., uniform sampled3D tex3;

- Some helper methods
- The OCIO shader function may be broken down as:
- The function header e.g., void OCIODisplay(in vec4 inColor) {
- The function body e.g., vec4 outColor.rgb = texture3D(tex3, inColor.rgb).rgb;
- The function footer e.g., return outColor; }

Usage Example:

Below is a code snippet to highlight the different parts of the OCIO shader program.

```
// All global declarations
uniform sampled3D tex3;

// All helper methods
vec3 computePosition(vec3 color)
{
   vec3 coords = color;
   // Some processing...
   return coords;
}

// The shader function
vec4 OCIODisplay(in vec4 inColor)     //
{                                     // Function Header
   vec4 outColor = inColor;           //

   outColor.rgb = texture3D(tex3, computePosition(inColor.rgb)).rgb;

   return outColor;                   // Function Footer
}                                     //
```

TEXTURE_1D = <TextureDimensions.TEXTURE_1D: 1>

TEXTURE_2D = <TextureDimensions.TEXTURE_2D: 2>

class TextureDimensions

Dimension enum used to differentiate between 1D and 2D object/resource types.

Members:

TEXTURE_1D

TEXTURE_2D

TEXTURE_1D = <TextureDimensions.TEXTURE_1D: 1>

TEXTURE_2D = <TextureDimensions.TEXTURE_2D: 2>

property name

property value

addToFunctionFooterShaderCode(*shaderCode: str*) → None

addToFunctionHeaderShaderCode(*shaderCode: str*) → None

addToFunctionShaderCode(*shaderCode: str*) → None

addToHelperShaderCode(*shaderCode: str*) → None

addToParameterDeclareShaderCode(*shaderCode: str*) → None

addToTextureDeclareShaderCode(*shaderCode: str*) → None

begin(*uid: str*) → None

Start to collect the shader data.

clone() → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator*

createShaderText(*shaderParameterDeclarations: str*, *shaderTextureDeclarations: str*, *shaderHelperMethods: str*, *shaderFunctionHeader: str*, *shaderFunctionBody: str*, *shaderFunctionFooter: str*) → None

Create the OCIO shader program.

Note: The OCIO shader program is decomposed to allow a specific implementation to change some parts. Some product integrations add the color processing within a client shader program, imposing constraints requiring this flexibility.

---

end() → None

End to collect the shader data.

finalize() → None

getAllowTexture1D() → bool

getCacheID() → str

getDescriptorSetIndex() → int

getDynamicProperties() → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator*

getDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

getFunctionName() → str

getLanguage() → *PyOpenColorIO.PyOpenColorIO.GpuLanguage*

getNextResourceIndex() → int

To avoid global texture sampler and uniform name clashes always append an increasing index to the resource name.

getPixelName() → str

getResourcePrefix() → str

Note: Some applications require that textures, uniforms, and helper methods be uniquely named because several processor instances could coexist.

---

getTextureBindingStart() → int

getTextureMaxWidth() → int

getUniqueID() → str

hasDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → bool

setAllowTexture1D(*allowed: bool*) → None

Allow 1D GPU resource type, otherwise always using 2D resources for 1D LUTs.

setDescriptorSetIndex(*index: SupportsInt*, *textureBindingStart: SupportsInt*) → None

Set the descriptor set index and texture binding start index to use for the shader program.

Note: Only supported for shading languages, such as Vulkan, that use descriptor sets and texture bindings.

Parameters

- **index** – The descriptor set index to use.
- **textureBindingStart** – The texture binding start index to use. The default index starts at 1 and is incremented by 1 for each texture. Otherwise, the texture binding starts at textureBindingStart and is incremented by 1 for each texture. The binding of a texture is equal to the texture index + textureBindingStart. The texture binding start index must be greater than 0, as binding 0 is reserved for the uniform buffer binding

setFunctionName(*name: str*) → None

setLanguage(*language:* PyOpenColorIO.PyOpenColorIO.GpuLanguage) → None

Set the shader program language.

setPixelName(*name: str*) → None

Set the pixel name variable holding the color values.

setResourcePrefix(*prefix: str*) → None

Set a prefix to the resource name.

setTextureMaxWidth(*maxWidth: SupportsInt*) → None

Some graphic cards could have 1D & 2D textures with size limitations.

setUniqueID(*uid: str*) → None

class PyOpenColorIO.GpuShaderCreator.**TextureType**

Members:

TEXTURE_RED_CHANNEL

TEXTURE_RGB_CHANNEL

TEXTURE_RED_CHANNEL = <TextureType.TEXTURE_RED_CHANNEL: 0>

TEXTURE_RGB_CHANNEL = <TextureType.TEXTURE_RGB_CHANNEL: 1>

property value

class PyOpenColorIO.GpuShaderCreator.**DynamicPropertyIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

C++

class **GpuShaderCreator**

Inherit from the class to fully customize the implementation of a GPU shader program from a color transformation.

When no customizations are needed and the intermediate in-memory step is acceptable then the GpuShaderDesc is a better choice.

An OCIO shader program could contain:

- A declaration part e.g., uniform sampled3D tex3;
- Some helper methods
- The OCIO shader function may be broken down as:
  - The function header e.g., void OCIODisplay(in vec4 inColor) {
  - The function body e.g., vec4 outColor.rgb = texture3D(tex3, inColor.rgb).rgb;
  - The function footer e.g., return outColor; }

Usage Example:

Below is a code snippet to highlight the different parts of the OCIO shader program.

```
// All global declarations
uniform sampled3D tex3;

// All helper methods
vec3 computePosition(vec3 color)
{
   vec3 coords = color;
   // Some processing...
   return coords;
}

// The shader function
vec4 OCIODisplay(in vec4 inColor)     //
{                                     // Function Header
   vec4 outColor = inColor;           //

   outColor.rgb = texture3D(tex3, computePosition(inColor.rgb)).rgb;

   return outColor;                   // Function Footer
}                                     //
```

Note: To better decouple the DynamicProperties from their GPU implementation, the code provides several addUniform() methods i.e. one per access function types. For example, an ExposureContrastTransform instance owns three DynamicProperties and they are all implemented by a double. When creating the GPU fragment

shader program, the addUniform() with GpuShaderCreator::DoubleGetter is called when property is dynamic, up to three times.

Subclassed by GpuShaderDesc

Public Types

enum **TextureType**

Values:

enumerator **TEXTURE_RED_CHANNEL**

Only need a red channel texture.

enumerator **TEXTURE_RGB_CHANNEL**

Need a RGB texture.

enum **TextureDimensions**

Dimension enum used to differentiate between 1D and 2D object/resource types.

Values:

enumerator **TEXTURE_1D**

enumerator **TEXTURE_2D**

typedef std::function<double()> **DoubleGetter**

Function returning a double, used by uniforms. GPU converts double to float.

typedef std::function<bool()> **BoolGetter**

Function returning a bool, used by uniforms.

typedef std::function<const Float3&()> **Float3Getter**

Functions returning a Float3, used by uniforms.

typedef std::function<int()> **SizeGetter**

Function returning an int, used by uniforms.

typedef std::function<const float*()> **VectorFloatGetter**

Function returning a float *, used by uniforms.

typedef std::function<const int*()> **VectorIntGetter**

Function returning an int *, used by uniforms.

Public Functions

virtual *GpuShaderCreatorRcPtr* **clone**() const = 0

const char ***getUniqueID**() const noexcept

void **setUniqueID**(const char *uid) noexcept

GpuLanguage **getLanguage**() const noexcept

void **setLanguage**(*GpuLanguage* lang) noexcept

Set the shader program language.

const char ***getFunctionName**() const noexcept

void **setFunctionName**(const char *name) noexcept

const char ***getPixelName**() const noexcept

void **setPixelName**(const char *name) noexcept

Set the pixel name variable holding the color values.

const char ***getResourcePrefix**() const noexcept

Note: Some applications require that textures, uniforms, and helper methods be uniquely named because several processor instances could coexist.

---

void **setResourcePrefix**(const char *prefix) noexcept

Set a prefix to the resource name.

void **setDescriptorSetIndex**(unsigned index, unsigned textureBindingStart = 1)

Set the descriptor set index and texture binding start index to use for the shader program.

Note: Only supported for shading languages, such as Vulkan, that use descriptor sets and texture bindings.

Parameters

- **index** – The descriptor set index to use.
- **textureBindingStart** – The texture binding start index to use. The default index starts at 1 and is incremented by 1 for each texture. Otherwise, the texture binding starts at textureBindingStart and is incremented by 1 for each texture. The binding of a texture is equal to the texture index + textureBindingStart. The texture binding start index must be greater than 0, as binding 0 is reserved for the uniform buffer binding

unsigned **getDescriptorSetIndex**() const noexcept

unsigned **getTextureBindingStart**() const noexcept

virtual const char ***getCacheID**() const noexcept

virtual void **begin**(const char *uid)

Start to collect the shader data.

virtual void **end**()

End to collect the shader data.

virtual void **setTextureMaxWidth**(unsigned maxWidth) = 0

Some graphic cards could have 1D & 2D textures with size limitations.

virtual unsigned **getTextureMaxWidth**() const noexcept = 0

virtual void **setAllowTexture1D**(bool allowed) = 0

Allow 1D GPU resource type, otherwise always using 2D resources for 1D LUTs.

virtual bool **getAllowTexture1D**() const = 0

unsigned **getNextResourceIndex**() noexcept

To avoid global texture sampler and uniform name clashes always append an increasing index to the resource name.

virtual bool **addUniform**(const char *name, const *DoubleGetter* &getDouble) = 0

virtual bool **addUniform**(const char *name, const *BoolGetter* &getBool) = 0

virtual bool **addUniform**(const char *name, const *Float3Getter* &getFloat3) = 0

virtual bool **addUniform**(const char *name, const *SizeGetter* &getSize, const *VectorFloatGetter* &getVectorFloat, const unsigned maxSize) = 0

The size of the vector can be smaller than the size of the corresponding array that is declared in the shader. The parameter maxSize must be used to pass the size of the array declared in the shader. This is important for being able to calculate the correct uniform buffer offset for subsequent uniforms

virtual bool **addUniform**(const char *name, const *SizeGetter* &getSize, const *VectorIntGetter* &getVectorInt, const unsigned maxSize) = 0

The size of the vector can be smaller than the size of the corresponding array that is declared in the shader. The parameter maxSize must be used to pass the size of the array declared in the shader. This is important for being able to calculate the correct uniform buffer offset for subsequent uniforms

void **addDynamicProperty**(*DynamicPropertyRcPtr* &prop)

Adds the property (used internally).

unsigned **getNumDynamicProperties**() const noexcept

Dynamic Property related methods.

DynamicPropertyRcPtr **getDynamicProperty**(unsigned index) const

bool **hasDynamicProperty**(*DynamicPropertyType* type) const

DynamicPropertyRcPtr **getDynamicProperty**(*DynamicPropertyType* type) const

Dynamic properties allow changes once the fragment shader program has been created. The steps are to get the appropriate DynamicProperty instance, and then change its value.

virtual unsigned **addTexture**(const char *textureName, const char *samplerName, unsigned width, unsigned height, *TextureType* channel, *TextureDimensions* dimensions, *Interpolation* interpolation, const float *values) = 0

Add a 1D or 2D texture

Note: The 'values' parameter contains the LUT data which must be used as-is as the dimensions and origin are hard-coded in the fragment shader program. So, it means one GPU texture per entry.

Returns
Shader binding index of the texture. For shading languages using explicit texture bindings, the return value is the same as the texture binding index in the generated shader program. The setDescriptorSetIndex function may be used to offset the starting index value.

virtual unsigned **add3DTexture**(const char *textureName, const char *samplerName, unsigned edgelen, *Interpolation* interpolation, const float *values) = 0

Add a 3D texture with RGB channel type.

Note: The 'values' parameter contains the 3D LUT data which must be used as-is as the dimension and origin are hard-coded in the fragment shader program. So, it means one GPU 3D texture per entry.

Returns
Shader binding index of the texture. For shading languages using explicit texture bindings, the return value is the same as the texture binding index in the generated shader program. The setDescriptorSetIndex function may be used to offset the starting index value.

virtual void **addToParameterDeclareShaderCode**(const char *shaderCode)

virtual void **addToTextureDeclareShaderCode**(const char *shaderCode)

virtual void **addToHelperShaderCode**(const char *shaderCode)

virtual void **addToFunctionHeaderShaderCode**(const char *shaderCode)

virtual void **addToFunctionShaderCode**(const char *shaderCode)

virtual void **addToFunctionFooterShaderCode**(const char *shaderCode)

virtual void **createShaderText**(const char *shaderParameterDeclarations, const char *shaderTextureDeclarations, const char *shaderHelperMethods, const char *shaderFunctionHeader, const char *shaderFunctionBody, const char *shaderFunctionFooter)

Create the OCIO shader program.

Note: The OCIO shader program is decomposed to allow a specific implementation to change some parts. Some product integrations add the color processing within a client shader program, imposing constraints requiring this flexibility.

---

virtual void **finalize**()

GpuShaderCreator(const *GpuShaderCreator*&) = delete

GpuShaderCreator &**operator=**(const *GpuShaderCreator*&) = delete

virtual **~GpuShaderCreator**()

Do not use (needed only for pybind11).

typedef std::shared_ptr<const *GpuShaderCreator*> OpenColorIO_v2_5::**ConstGpuShaderCreatorRcPtr**

typedef std::shared_ptr<*GpuShaderCreator*> OpenColorIO_v2_5::**GpuShaderCreatorRcPtr**

GpuShaderDesc

Python

class PyOpenColorIO.**GpuShaderDesc**

This class holds the GPU-related information needed to build a shader program from a specific processor.

This class defines the interface and there are two implementations provided. The "legacy" mode implements the OCIO v1 approach of baking certain ops in order to have at most one 3D-LUT. The "generic" mode is the v2 default and allows all the ops to be processed as-is, without baking, like the CPU renderer. Custom implementations could be written to accommodate the GPU needs of a specific client app.

The complete fragment shader program is decomposed in two main parts: the OCIO shader program for the color processing and the client shader program which consumes the pixel color processing.

The OCIO shader program is fully described by the GpuShaderDesc independently from the client shader program. The only critical point is the agreement on the OCIO function shader name.

To summarize, the complete shader program is:

```
//                                                                  //
//              The complete fragment shader program                //
//                                                                  //
//                                                                  //
//  ////////////////////////////////////////////////////////////    //
//  //                                                        //    //
//  //              The OCIO shader program                   //    //
//  //                                                        //    //
//  ////////////////////////////////////////////////////////////    //
//  //                                                        //    //
//  //  // All global declarations                           //    //
//  //  uniform sampled3D tex3;                               //    //
//  //                                                        //    //
//  //  // All helper methods                                 //    //
//  //  vec3 computePos(vec3 color)                           //    //
//  //  {                                                     //    //
//  //     vec3 coords = color;                               //    //
//  //     ...                                                //    //
//  //     return coords;                                     //    //
//  //  }                                                     //    //
//  //                                                        //    //
//  //  // The OCIO shader function                           //    //
//  //  vec4 OCIODisplay(in vec4 inColor)                     //    //
//  //  {                                                     //    //
//  //     vec4 outColor = inColor;                           //    //
//  //     ...                                                //    //
//  //     outColor.rbg                                       //    //
//  //        = texture3D(tex3, computePos(inColor.rgb)).rgb; //    //
//  //     ...                                                //    //
//  //     return outColor;                                   //    //
//  //  }                                                     //    //
//  //                                                        //    //
//  ////////////////////////////////////////////////////////////    //
//                                                                  //
//  ////////////////////////////////////////////////////////////    //
//  //                                                        //    //
```

(continues on next page)

(continued from previous page)

```
//   //            The client shader program                      //   //
//   //                                                           //   //
//   ///////////////////////////////////////////////////////////////   //
//   //                                                           //   //
//   //   uniform sampler2D image;                                //   //
//   //                                                           //   //
//   //   void main()                                             //   //
//   //   {                                                       //   //
//   //      vec4 inColor = texture2D(image, gl_TexCoord[0].st); //   //
//   //      ...                                                  //   //
//   //      vec4 outColor = OCIODisplay(inColor);                //   //
//   //      ...                                                  //   //
//   //      gl_FragColor = outColor;                             //   //
//   //   }                                                       //   //
//   //                                                           //   //
//   ///////////////////////////////////////////////////////////////   //
//                                                                     //
/////////////////////////////////////////////////////////////////////////
```

Usage Example: *Building a GPU shader*

This example is based on the code in: src/apps/ociodisplay/main.cpp

```
// Get the processor
//
OCIO::ConstConfigRcPtr config = OCIO::Config::CreateFromEnv();
OCIO::ConstProcessorRcPtr processor
   = config->getProcessor("ACES - ACEScg", "Output - sRGB");

// Step 1: Create a GPU shader description
//
// The two potential scenarios are:
//
//   1. Instantiate the generic shader description.  The color processor
//      is used as-is (i.e. without any baking step) and could contain
//      any number of 1D & 3D luts.
//
//      This is the default OCIO v2 behavior and allows a much better
//      match between the CPU and GPU renderers.
//
OCIO::GpuShaderDescRcPtr shaderDesc = OCIO::GpuShaderDesc::Create();
//
//   2. Instantiate a custom shader description.
//
//      Writing a custom shader description is a way to tailor the shaders
//      to the needs of a given client program.  This involves writing a
//      new class inheriting from the pure virtual class GpuShaderDesc.
//
//      Please refer to the GenericGpuShaderDesc class for an example.
//
OCIO::GpuShaderDescRcPtr shaderDesc = MyCustomGpuShader::Create();

shaderDesc->setLanguage(OCIO::GPU_LANGUAGE_GLSL_1_2);
```

(continues on next page)

(continued from previous page)

```
shaderDesc->setFunctionName("OCIODisplay");

// Step 2: Collect the shader program information for a specific processor
//
processor->extractGpuShaderInfo(shaderDesc);

// Step 3: Create a helper to build the shader. Here we use a helper for
//         OpenGL but there will also be helpers for other languages.
//
OpenGLBuilderRcPtr oglBuilder = OpenGLBuilder::Create(shaderDesc);

// Step 4: Allocate & upload all the LUTs
//
oglBuilder->allocateAllTextures();

// Step 5: Build the complete fragment shader program using
//         g_fragShaderText which is the client shader program.
//
g_programId = oglBuilder->buildProgram(g_fragShaderText);

// Step 6: Enable the fragment shader program, and all needed textures
//
glUseProgram(g_programId);
glUniform1i(glGetUniformLocation(g_programId, "tex1"), 1);  // image texture
oglBuilder->useAllTextures(g_programId);                    // LUT textures

// Step 7: Update uniforms from dynamic property instances.
m_oglBuilder->useAllUniforms();
```

static CreateShaderDesc(*language: PyOpenColorIO.PyOpenColorIO.GpuLanguage = <GpuLanguage.GPU_LANGUAGE_GLSL_1_2: 1>*, *functionName: str = 'OCIOMain'*, *pixelName: str = 'outColor'*, *resourcePrefix: str = 'ocio'*, *uid: str = ''*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc*

Create the default shader description.

TEXTURE_1D = <TextureDimensions.TEXTURE_1D: 1>

TEXTURE_2D = <TextureDimensions.TEXTURE_2D: 2>

class TextureDimensions

Dimension enum used to differentiate between 1D and 2D object/resource types.

Members:

TEXTURE_1D

TEXTURE_2D

TEXTURE_1D = <TextureDimensions.TEXTURE_1D: 1>

TEXTURE_2D = <TextureDimensions.TEXTURE_2D: 2>

TextureDimensions(*value: SupportsInt*) → None

property name

property value

TextureDimensions(*args*, ***kwargs*)

add3DTexture(*textureName: str*, *samplerName: str*, *edgeLen: SupportsInt*, *interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation, *values: typing_extensions.Buffer*) → int

Add a 3D texture with RGB channel type.

Note: The 'values' parameter contains the 3D LUT data which must be used as-is as the dimension and origin are hard-coded in the fragment shader program. So, it means one GPU 3D texture per entry.

Returns

Shader binding index of the texture. For shading languages using explicit texture bindings, the return value is the same as the texture binding index in the generated shader program. The setDescriptorSetIndex function may be used to offset the starting index value.

addTexture(*textureName: str*, *samplerName: str*, *width: SupportsInt*, *height: SupportsInt*, *channel:* PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.TextureType, *dimensions:* PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.TextureDimensions, *interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation, *values: typing_extensions.Buffer*) → int

Add a 1D or 2D texture

Note: The 'values' parameter contains the LUT data which must be used as-is as the dimensions and origin are hard-coded in the fragment shader program. So, it means one GPU texture per entry.

Returns

Shader binding index of the texture. For shading languages using explicit texture bindings, the return value is the same as the texture binding index in the generated shader program. The setDescriptorSetIndex function may be used to offset the starting index value.

addToFunctionFooterShaderCode(*shaderCode: str*) → None

addToFunctionHeaderShaderCode(*shaderCode: str*) → None

addToFunctionShaderCode(*shaderCode: str*) → None

addToHelperShaderCode(*shaderCode: str*) → None

addToParameterDeclareShaderCode(*shaderCode: str*) → None

addToTextureDeclareShaderCode(*shaderCode: str*) → None

begin(*uid: str*) → None

Start to collect the shader data.

clone() → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator*

createShaderText(*shaderParameterDeclarations: str*, *shaderTextureDeclarations: str*, *shaderHelperMethods: str*, *shaderFunctionHeader: str*, *shaderFunctionBody: str*, *shaderFunctionFooter: str*) → None

Create the OCIO shader program.

Note: The OCIO shader program is decomposed to allow a specific implementation to change some parts. Some product integrations add the color processing within a client shader program, imposing constraints requiring this flexibility.

end() → None

End to collect the shader data.

finalize() → None

get3DTextures() → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture3DIterator*

getAllowTexture1D() → bool

getCacheID() → str

getDescriptorSetIndex() → int

getDynamicProperties() → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator*

getDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

getFunctionName() → str

getLanguage() → *PyOpenColorIO.PyOpenColorIO.GpuLanguage*

getNextResourceIndex() → int

To avoid global texture sampler and uniform name clashes always append an increasing index to the resource name.

getPixelName() → str

getResourcePrefix() → str

Note: Some applications require that textures, uniforms, and helper methods be uniquely named because several processor instances could coexist.

getShaderText() → str

Get the complete OCIO shader program.

getTextureBindingStart() → int

getTextureMaxWidth() → int

getTextures() → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.TextureIterator*

getUniformBufferSize() → int

For shading languages that use uniform buffers, a uniform buffer containing all uniforms is generated in the shader code. This method can be used to create a buffer of the same size in the client code that can be filled with the corresponding data.

Returns

Size of the uniform buffer in bytes

getUniforms() → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.UniformIterator*

getUniqueID() → str

hasDynamicProperty(*type:* PyOpenColorIO.PyOpenColorIO.DynamicPropertyType) → bool

setAllowTexture1D(*allowed: bool*) → None

Allow 1D GPU resource type, otherwise always using 2D resources for 1D LUTs.

setDescriptorSetIndex(*index: SupportsInt*, *textureBindingStart: SupportsInt*) → None

Set the descriptor set index and texture binding start index to use for the shader program.

Note: Only supported for shading languages, such as Vulkan, that use descriptor sets and texture bindings.

Parameters

- **index** – The descriptor set index to use.
- **textureBindingStart** – The texture binding start index to use. The default index starts at 1 and is incremented by 1 for each texture. Otherwise, the texture binding starts at textureBindingStart and is incremented by 1 for each texture. The binding of a texture is equal to the texture index + textureBindingStart. The texture binding start index must be greater than 0, as binding 0 is reserved for the uniform buffer binding

setFunctionName(*name: str*) → None

setLanguage(*language:* PyOpenColorIO.PyOpenColorIO.GpuLanguage) → None

Set the shader program language.

setPixelName(*name: str*) → None

Set the pixel name variable holding the color values.

setResourcePrefix(*prefix: str*) → None

Set a prefix to the resource name.

setTextureMaxWidth(*maxWidth: SupportsInt*) → None

Some graphic cards could have 1D & 2D textures with size limitations.

setUniqueID(*uid: str*) → None

class PyOpenColorIO.GpuShaderDesc.**TextureType**

Members:

TEXTURE_RED_CHANNEL

TEXTURE_RGB_CHANNEL

TEXTURE_RED_CHANNEL = <TextureType.TEXTURE_RED_CHANNEL: 0>

TEXTURE_RGB_CHANNEL = <TextureType.TEXTURE_RGB_CHANNEL: 1>

property value

class PyOpenColorIO.GpuShaderDesc.**UniformData**

property bufferOffset

getBool() → bool

getDouble() → float

getFloat3() → Annotated[list[float], 'FixedSize(3)']

getVectorFloat() → numpy.ndarray

getVectorInt() → numpy.ndarray

property type

class PyOpenColorIO.GpuShaderDesc.**Texture**

property channel

property dimensions

getValues() → numpy.ndarray

property height

property interpolation

property samplerName

property textureName

property textureShaderBindingIndex

property width

class PyOpenColorIO.GpuShaderDesc.**Texture3D**

property edgeLen

getValues() → numpy.ndarray

property interpolation

property samplerName

property textureName

property textureShaderBindingIndex

class PyOpenColorIO.GpuShaderDesc.**UniformIterator**

self[*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.UniformIterator*

len(*self*) → int

next(*self*) → tuple

class PyOpenColorIO.GpuShaderDesc.**TextureIterator**

self[*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.TextureIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture*

class PyOpenColorIO.GpuShaderDesc.**Texture3DIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture3D*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture3DIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderDesc.Texture3D*

class PyOpenColorIO.GpuShaderDesc.**DynamicPropertyIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.DynamicProperty*

C++

class **GpuShaderDesc** : public *GpuShaderCreator*

This class holds the GPU-related information needed to build a shader program from a specific processor.

This class defines the interface and there are two implementations provided. The "legacy" mode implements the OCIO v1 approach of baking certain ops in order to have at most one 3D-LUT. The "generic" mode is the v2 default and allows all the ops to be processed as-is, without baking, like the CPU renderer. Custom implementations could be written to accommodate the GPU needs of a specific client app.

The complete fragment shader program is decomposed in two main parts: the OCIO shader program for the color processing and the client shader program which consumes the pixel color processing.

The OCIO shader program is fully described by the GpuShaderDesc independently from the client shader program. The only critical point is the agreement on the OCIO function shader name.

To summarize, the complete shader program is:

```
//                                                                  //
//              The complete fragment shader program                //
//                                                                  //
//                                                                  //
//   ////////////////////////////////////////////////////////////   //
//   //                                                        //   //
//   //              The OCIO shader program                   //   //
//   //                                                        //   //
//   ////////////////////////////////////////////////////////////   //
//   //                                                        //   //
//   //   // All global declarations                          //   //
//   //   uniform sampled3D tex3;                             //   //
//   //                                                       //   //
//   //   // All helper methods                               //   //
//   //   vec3 computePos(vec3 color)                         //   //
//   //   {                                                   //   //
```

(continues on next page)

(continued from previous page)

```
//   //      vec3 coords = color;                                    //   //
//   //      ...                                                     //   //
//   //      return coords;                                          //   //
//   //   }                                                          //   //
//   //                                                              //   //
//   //   // The OCIO shader function                               //   //
//   //   vec4 OCIODisplay(in vec4 inColor)                          //   //
//   //   {                                                          //   //
//   //      vec4 outColor = inColor;                                //   //
//   //      ...                                                     //   //
//   //      outColor.rbg                                            //   //
//   //         = texture3D(tex3, computePos(inColor.rgb)).rgb;      //   //
//   //      ...                                                     //   //
//   //      return outColor;                                        //   //
//   //   }                                                          //   //
//   //                                                              //   //
//   ////////////////////////////////////////////////////////////////   //
//                                                                      //
//   ////////////////////////////////////////////////////////////////   //
//   //                                                            //   //
//   //             The client shader program                      //   //
//   //                                                            //   //
//   ////////////////////////////////////////////////////////////////   //
//   //                                                            //   //
//   //   uniform sampler2D image;                                 //   //
//   //                                                            //   //
//   //   void main()                                              //   //
//   //   {                                                        //   //
//   //      vec4 inColor = texture2D(image, gl_TexCoord[0].st);   //   //
//   //      ...                                                   //   //
//   //      vec4 outColor = OCIODisplay(inColor);                 //   //
//   //      ...                                                   //   //
//   //      gl_FragColor = outColor;                              //   //
//   //   }                                                        //   //
//   //                                                            //   //
//   ////////////////////////////////////////////////////////////////   //
//                                                                      //
////////////////////////////////////////////////////////////////////////
```

Usage Example: *Building a GPU shader*

This example is based on the code in: src/apps/ociodisplay/main.cpp

```
// Get the processor
//
OCIO::ConstConfigRcPtr config = OCIO::Config::CreateFromEnv();
OCIO::ConstProcessorRcPtr processor
   = config->getProcessor("ACES - ACEScg", "Output - sRGB");

// Step 1: Create a GPU shader description
//
// The two potential scenarios are:
//
```

(continues on next page)

(continued from previous page)

```
//   1. Instantiate the generic shader description.  The color processor
//      is used as-is (i.e. without any baking step) and could contain
//      any number of 1D & 3D luts.
//
//      This is the default OCIO v2 behavior and allows a much better
//      match between the CPU and GPU renderers.
//
OCIO::GpuShaderDescRcPtr shaderDesc = OCIO::GpuShaderDesc::Create();
//
//   2. Instantiate a custom shader description.
//
//      Writing a custom shader description is a way to tailor the shaders
//      to the needs of a given client program.  This involves writing a
//      new class inheriting from the pure virtual class GpuShaderDesc.
//
//      Please refer to the GenericGpuShaderDesc class for an example.
//
OCIO::GpuShaderDescRcPtr shaderDesc = MyCustomGpuShader::Create();

shaderDesc->setLanguage(OCIO::GPU_LANGUAGE_GLSL_1_2);
shaderDesc->setFunctionName("OCIODisplay");

// Step 2: Collect the shader program information for a specific processor
//
processor->extractGpuShaderInfo(shaderDesc);

// Step 3: Create a helper to build the shader. Here we use a helper for
//         OpenGL but there will also be helpers for other languages.
//
OpenGLBuilderRcPtr oglBuilder = OpenGLBuilder::Create(shaderDesc);

// Step 4: Allocate & upload all the LUTs
//
oglBuilder->allocateAllTextures();

// Step 5: Build the complete fragment shader program using
//         g_fragShaderText which is the client shader program.
//
g_programId = oglBuilder->buildProgram(g_fragShaderText);

// Step 6: Enable the fragment shader program, and all needed textures
//
glUseProgram(g_programId);
glUniform1i(glGetUniformLocation(g_programId, "tex1"), 1);  // image texture
oglBuilder->useAllTextures(g_programId);                    // LUT textures

// Step 7: Update uniforms from dynamic property instances.
m_oglBuilder->useAllUniforms();
```

Public Functions

virtual *GpuShaderCreatorRcPtr* **clone**() const override

virtual unsigned **getNumUniforms**() const noexcept = 0

virtual const char ***getUniform**(unsigned index, *UniformData* &data) const = 0

Returns name of uniform and data as parameter.

virtual std::size_t **getUniformBufferSize**() const noexcept = 0

For shading languages that use uniform buffers, a uniform buffer containing all uniforms is generated in the shader code. This method can be used to create a buffer of the same size in the client code that can be filled with the corresponding data.

Returns
Size of the uniform buffer in bytes

virtual unsigned **getNumTextures**() const noexcept = 0

The getTexture methods are used to access Lut1D arrays to upload to the GPU as textures. Please note that the index used here is based on the total number of Lut1Ds used by the Processor and is different from the texture shader binding index, which may be obtained using the corresponding function.

virtual void **getTexture**(unsigned index, const char *&textureName, const char *&samplerName, unsigned &width, unsigned &height, TextureType &channel, TextureDimensions &dimensions, *Interpolation* &interpolation) const = 0

virtual void **getTextureValues**(unsigned index, const float *&values) const = 0

virtual unsigned **getTextureShaderBindingIndex**(unsigned index) const = 0

Get the index used to declare the texture in the shader for languages such as Vulkan.

virtual unsigned **getNum3DTextures**() const noexcept = 0

The get3DTexture methods are used to access Lut3D arrays to upload to the GPU as textures. Please note that the index used here is based on the total number of Lut3Ds used by the Processor and is different from the texture shader binding index, which may be obtained using the corresponding function.

virtual void **get3DTexture**(unsigned index, const char *&textureName, const char *&samplerName, unsigned &edgelen, *Interpolation* &interpolation) const = 0

virtual void **get3DTextureValues**(unsigned index, const float *&values) const = 0

virtual unsigned **get3DTextureShaderBindingIndex**(unsigned index) const = 0

Get the index used to declare the texture in the shader for languages such as Vulkan.

const char ***getShaderText**() const noexcept

Get the complete OCIO shader program.

GpuShaderDesc(const *GpuShaderDesc*&) = delete

GpuShaderDesc &**operator=**(const *GpuShaderDesc*&) = delete

virtual **~GpuShaderDesc**()

Do not use (needed only for pybind11).

Public Static Functions

static *GpuShaderDescRcPtr* **CreateShaderDesc**()

Create the default shader description.

struct **UniformData**

Used to retrieve uniform information. UniformData m_type indicates the type of uniform and what member of the structure should be used:

- UNIFORM_DOUBLE: m_getDouble.
- UNIFORM_BOOL: m_getBool.
- UNIFORM_FLOAT3: m_getFloat3.
- UNIFORM_VECTOR_FLOAT: m_vectorFloat.
- UNIFORM_VECTOR_INT: m_vectorInt.

The m_bufferOffset is the offset in bytes from the start of the uniform buffer. For shading languages that use uniform buffers, the offset can be used to determine the location of the uniform in the buffer and fill it with the corresponding data.

Public Members

UniformDataType **m_type** = {UNIFORM_UNKNOWN}

std::size_t **m_bufferOffset** = { }

DoubleGetter **m_getDouble** = { }

BoolGetter **m_getBool** = { }

Float3Getter **m_getFloat3** = { }

struct OpenColorIO_v2_5::*GpuShaderDesc*::*UniformData*::*VectorFloat* **m_vectorFloat**

struct OpenColorIO_v2_5::*GpuShaderDesc*::*UniformData*::*VectorInt* **m_vectorInt**

struct **VectorFloat**

Public Members

SizeGetter **m_getSize** = { }

VectorFloatGetter **m_getVector** = { }

struct **VectorInt**

Public Members

SizeGetter **m_getSize** = { }

VectorIntGetter **m_getVector** = { }

typedef std::shared_ptr<const *GpuShaderDesc*> OpenColorIO_v2_5::**ConstGpuShaderDescRcPtr**

typedef std::shared_ptr<*GpuShaderDesc*> OpenColorIO_v2_5::**GpuShaderDescRcPtr**

5.7.25 SystemMonitors

Python

class PyOpenColorIO.**SystemMonitors**

Provides access to the ICC monitor profile provided by the operating system for each active display.

SystemMonitors → None

getMonitors() → *PyOpenColorIO.PyOpenColorIO.SystemMonitors.MonitorIterator*

class PyOpenColorIO.SystemMonitors.**MonitorIterator**

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.SystemMonitors.MonitorIterator*

len(*self*) → int

next(*self*) → tuple

C++

class **SystemMonitors**

Provides access to the ICC monitor profile provided by the operating system for each active display.

Public Functions

SystemMonitors(const *SystemMonitors&*) = delete

SystemMonitors &**operator=**(const *SystemMonitors&*) = delete

virtual bool **isSupported**() const noexcept = 0

True if the OS is able to provide ICC profiles for the attached monitors (macOS, Windows) and false otherwise.

virtual size_t **getNumMonitors**() const noexcept = 0

Get the number of active monitors reported by the operating system.

Methods to access some information of the attached and active monitors.

virtual const char ***getMonitorName**(size_t idx) const = 0

Get the monitor profile name.

Get the string describing the monitor. It is used as an argument to instantiateDisplay. It may also be used in a UI to ask a user which of several monitors they want to instantiate a display for.

virtual const char ***getProfileFilepath**(size_t idx) const = 0

Get the ICC profile path associated to the monitor.

Public Static Functions

static *ConstSystemMonitorsRcPtr* **Get**() noexcept

Get the existing instance.

typedef std::shared_ptr<const *SystemMonitors*> OpenColorIO_v2_5::**ConstSystemMonitorsRcPtr**

typedef std::shared_ptr<*SystemMonitors*> OpenColorIO_v2_5::**SystemMonitorsRcPtr**

5.7.26 Transforms

See also: *Grading Transforms*

Transform

Python

class PyOpenColorIO.**Transform**

Base class for all the transform classes.

Transform(**args*, ***kwargs*)

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

validate() → None

Will throw if data is not valid.

C++

class **Transform**

Base class for all the transform classes.

Subclassed by AllocationTransform, BuiltinTransform, CDLTransform, ColorSpaceTransform, DisplayViewTransform, ExponentTransform, ExponentWithLinearTransform, ExposureContrastTransform, FileTransform, FixedFunctionTransform, GradingHueCurveTransform, GradingPrimaryTransform, GradingRGBCurveTransform, GradingToneTransform, GroupTransform, LogAffineTransform, LogCameraTransform, LogTransform, LookTransform, Lut1DTransform, Lut3DTransform, MatrixTransform, RangeTransform

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const = 0

virtual *TransformDirection* **getDirection**() const noexcept = 0

virtual void **setDirection**(*TransformDirection* dir) noexcept = 0

Note that this only affects the evaluation and not the values stored in the object.

virtual *TransformType* **getTransformType**() const noexcept = 0

virtual void **validate**() const

Will throw if data is not valid.

Transform(const *Transform*&) = delete

Transform &**operator=**(const *Transform*&) = delete

virtual **~Transform**() = default

Do not use (needed only for pybind11).

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *Transform*&)

typedef std::shared_ptr<const *Transform*> OpenColorIO_v2_5::**ConstTransformRcPtr**

typedef std::shared_ptr<*Transform*> OpenColorIO_v2_5::**TransformRcPtr**

AllocationTransform

Python

class PyOpenColorIO.**AllocationTransform**

Forward direction wraps the 'expanded' range into the specified, often compressed, range.

AllocationTransform(*args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.AllocationTransform) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.AllocationTransform, allocation: PyOpenColorIO.PyOpenColorIO.Allocation = <Allocation.ALLOCATION_UNIFORM: 1>, vars: collections.abc.Sequence[typing.SupportsFloat] = [], direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

getAllocation() → *PyOpenColorIO.PyOpenColorIO.Allocation*

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getVars() → list[float]

setAllocation(*allocation:* PyOpenColorIO.PyOpenColorIO.Allocation) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setVars(*vars: collections.abc.Sequence[SupportsFloat]*) → None

validate() → None

Will throw if data is not valid.

C++

class **AllocationTransform** : public *Transform*

Forward direction wraps the 'expanded' range into the specified, often compressed, range.

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const override

virtual *TransformDirection* **getDirection**() const noexcept override

virtual void **setDirection**(*TransformDirection* dir) noexcept override

Note that this only affects the evaluation and not the values stored in the object.

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual void **validate**() const override

Will throw if data is not valid.

Allocation **getAllocation**() const

void **setAllocation**(*Allocation* allocation)

int **getNumVars**() const

void **getVars**(float *vars) const

void **setVars**(int numvars, const float *vars)

AllocationTransform &**operator=**(const *AllocationTransform*&) = delete

virtual **~AllocationTransform**()

Do not use (needed only for pybind11).

Public Static Functions

static *AllocationTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *AllocationTransform*&)

typedef std::shared_ptr<const *AllocationTransform*> OpenColorIO_v2_5::**ConstAllocationTransformRcPtr**

typedef std::shared_ptr<*AllocationTransform*> OpenColorIO_v2_5::**AllocationTransformRcPtr**

BuiltinTransform

Python

class PyOpenColorIO.**BuiltinTransform**

A built-in transform is similar to a FileTransform, but without the file. OCIO knows how to build a set of commonly used transforms on-demand, thus avoiding the need for external files and simplifying config authoring.

BuiltinTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.BuiltinTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.BuiltinTransform, style: str = 'IDENTITY', direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

getDescription() → str

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getStyle() → str

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setStyle(*style: str*) → None

Select an existing built-in transform style from the list accessible through :ref:`BuiltinTransformRegistry`. The style is the ID string that identifies which transform to apply.

validate() → None

Will throw if data is not valid.

C++

class **BuiltinTransform** : public *Transform*

A built-in transform is similar to a FileTransform, but without the file. OCIO knows how to build a set of commonly used transforms on-demand, thus avoiding the need for external files and simplifying config authoring.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const char ***getStyle**() const noexcept = 0

virtual void **setStyle**(const char *style) = 0

Select an existing built-in transform style from the list accessible through :cpp:class:`BuiltinTransformRegistry`. The style is the ID string that identifies which transform to apply.

virtual const char ***getDescription**() const noexcept = 0

virtual **~BuiltinTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *BuiltinTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *BuiltinTransform*&) noexcept

typedef std::shared_ptr<const *BuiltinTransform*> OpenColorIO_v2_5::**ConstBuiltinTransformRcPtr**

typedef std::shared_ptr<*BuiltinTransform*> OpenColorIO_v2_5::**BuiltinTransformRcPtr**

BuiltinTransformRegistry

Python

class PyOpenColorIO.**BuiltinTransformRegistry**

The built-in transform registry contains all the existing built-in transforms which can be used by a configuration (version 2 or higher only).

BuiltinTransformRegistry → None

Get the current built-in transform registry.

getBuiltins() → *PyOpenColorIO.PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator*

class PyOpenColorIO.BuiltinTransformRegistry.**BuiltinStyleIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.BuiltinTransformRegistry.BuiltinStyleIterator*

len(*self*) → int

next(*self*) → str

class PyOpenColorIO.BuiltinTransformRegistry.**BuiltinIterator**

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator*

len(*self*) → int

next(*self*) → tuple

C++

class **BuiltinTransformRegistry**

The built-in transform registry contains all the existing built-in transforms which can be used by a configuration (version 2 or higher only).

Public Functions

BuiltinTransformRegistry(const *BuiltinTransformRegistry*&) = delete

BuiltinTransformRegistry &**operator=**(const *BuiltinTransformRegistry*&) = delete

virtual size_t **getNumBuiltins**() const noexcept = 0

Get the number of built-in transforms available.

virtual const char ***getBuiltinStyle**(size_t index) const = 0

Get the style string for the i-th built-in transform. The style is the ID string that identifies a given transform.

virtual const char ***getBuiltinDescription**(size_t index) const = 0

Get the description string for the i-th built-in transform.

Public Static Functions

static *ConstBuiltinTransformRegistryRcPtr* **Get**() noexcept

Get the current built-in transform registry.

typedef std::shared_ptr<const *BuiltinTransformRegistry*>
OpenColorIO_v2_5::**ConstBuiltinTransformRegistryRcPtr**

typedef std::shared_ptr<*BuiltinTransformRegistry*> OpenColorIO_v2_5::**BuiltinTransformRegistryRcPtr**

CDLTransform

Python

class PyOpenColorIO.**CDLTransform**

An implementation of the ASC Color Decision List (CDL), based on the ASC v1.2 specification.

ASC_SOP

Slope, offset, power:: out = clamp((in * slope) + offset) ^ power

---

Note: If the config version is 1, negative values are clamped if the power is not 1.0. For config version 2 and higher, the negative handling is controlled by the CDL style.

---

static CreateFromFile(*src: str*, *id: str*) → *PyOpenColorIO.PyOpenColorIO.CDLTransform*

Load the CDL from the src .cdl, .cc, or .ccc file.

Note: The cccid can be the ID of a CDL or the index of the CDL (as string). If cccid is NULL or empty the first CDL is returned. The cccid is case-sensitive. The src must be an absolute path reference, no relative directory or envvar resolution is performed. Throws if file does not contain any CDL or if the specified cccid is not found.

---

static CreateGroupFromFile(*src: str*) → *PyOpenColorIO.PyOpenColorIO.GroupTransform*

Load all of the CDLs in a .cdl or .ccc file into a single GroupTransform.

Note: This may be useful as a quicker way for applications to check the contents of each of the CDLs. The src must be an absolute path reference, no relative directory or envvar resolution is performed.

CDLTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.CDLTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.CDLTransform, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None
3. __init__(self: PyOpenColorIO.PyOpenColorIO.CDLTransform, slope: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [1.0, 1.0, 1.0], offset: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [0.0, 0.0, 0.0], power: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [1.0, 1.0, 1.0], sat: typing.SupportsFloat = 1.0, id: str = '', description: str = '', direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.CDLTransform) → bool

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFirstSOPDescription() → str

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getID() → str

The get/setID methods are now deprecated. The preferred way of interacting with the ID is now via the transform's formatMetadata.

getOffset() → Annotated[list[float], 'FixedSize(3)']

getPower() → Annotated[list[float], 'FixedSize(3)']

getSOP() → Annotated[list[float], 'FixedSize(9)']

getSat() → float

getSatLumaCoefs() → Annotated[list[float], 'FixedSize(3)']

These are hard-coded, by spec, to r709.

getSlope() → Annotated[list[float], 'FixedSize(3)']

getStyle() → *PyOpenColorIO.PyOpenColorIO.CDLStyle*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setFirstSOPDescription(*description: str*) → None

setID(*id: str*) → None

setOffset(*rgb: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setPower(*rgb: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setSOP(*vec9: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(9)']*) → None

setSat(*sat: SupportsFloat*) → None

setSlope(*rgb: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.CDLStyle) → None

Use CDL_ASC to clamp values to [0,1] per the ASC spec. Use NO_CLAMP to never clamp values (regardless of whether power is 1.0). The NO_CLAMP option passes negatives through unchanged (like the NEGATIVE_PASS_THRU style of ExponentTransform). The default style is CDL_NO_CLAMP.

validate() → None

Will throw if data is not valid.

C++

class **CDLTransform** : public *Transform*

An implementation of the ASC Color Decision List (CDL), based on the ASC v1.2 specification.

ASC_SOP

Slope, offset, power:: out = clamp((in * slope) + offset) ^ power

---

Note: If the config version is 1, negative values are clamped if the power is not 1.0. For config version 2 and higher, the negative handling is controlled by the CDL style.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual bool **equals**(const *CDLTransform* &other) const noexcept = 0

virtual *CDLStyle* **getStyle**() const = 0

virtual void **setStyle**(*CDLStyle* style) = 0

Use CDL_ASC to clamp values to [0,1] per the ASC spec. Use NO_CLAMP to never clamp values (regardless of whether power is 1.0). The NO_CLAMP option passes negatives through unchanged (like the NEGATIVE_PASS_THRU style of ExponentTransform). The default style is CDL_NO_CLAMP.

virtual void **getSlope**(double *rgb) const = 0

virtual void **setSlope**(const double *rgb) = 0

virtual void **getOffset**(double *rgb) const = 0

virtual void **setOffset**(const double *rgb) = 0

virtual void **getPower**(double *rgb) const = 0

virtual void **setPower**(const double *rgb) = 0

virtual void **getSOP**(double *vec9) const = 0

virtual void **setSOP**(const double *vec9) = 0

virtual double **getSat**() const = 0

virtual void **setSat**(double sat) = 0

virtual void **getSatLumaCoefs**(double *rgb) const = 0

These are hard-coded, by spec, to r709.

virtual const char ***getID**() const = 0

The get/setID methods are now deprecated. The preferred way of interacting with the ID is now via the transform's formatMetadata.

virtual void **setID**(const char *id) = 0

virtual const char ***getFirstSOPDescription**() const = 0

virtual void **setFirstSOPDescription**(const char *description) = 0

CDLTransform(const *CDLTransform*&) = delete

CDLTransform &**operator=**(const *CDLTransform*&) = delete

virtual **~CDLTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *CDLTransformRcPtr* **Create**()

static *CDLTransformRcPtr* **CreateFromFile**(const char *src, const char *cccid)

Load the CDL from the src .cdl, .cc, or .ccc file.

Note: The cccid can be the ID of a CDL or the index of the CDL (as string). If cccid is NULL or empty the first CDL is returned. The cccid is case-sensitive. The src must be an absolute path reference, no relative directory or envvar resolution is performed. Throws if file does not contain any CDL or if the specified cccid is not found.

---

static *GroupTransformRcPtr* **CreateGroupFromFile**(const char *src)

Load all of the CDLs in a .cdl or .ccc file into a single GroupTransform.

Note: This may be useful as a quicker way for applications to check the contents of each of the CDLs. The src must be an absolute path reference, no relative directory or envvar resolution is performed.

---

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *CDLTransform*&)

typedef std::shared_ptr<const *CDLTransform*> OpenColorIO_v2_5::**ConstCDLTransformRcPtr**

typedef std::shared_ptr<*CDLTransform*> OpenColorIO_v2_5::**CDLTransformRcPtr**

ColorSpaceTransform

Python

class PyOpenColorIO.**ColorSpaceTransform**

ColorSpaceTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpaceTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.ColorSpaceTransform, src: str = '', dst: str = '', direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>, dataBypass: bool = True) -> None

getDataBypass() → bool

Data color spaces do not get processed when true (which is the default).

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getDst() → str

getSrc() → str

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDataBypass(*dataBypass: bool*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setDst(*dst: str*) → None

setSrc(*src: str*) → None

validate() → None

Will throw if data is not valid.

C++

class **ColorSpaceTransform** : public *Transform*

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const override

virtual *TransformDirection* **getDirection**() const noexcept override

virtual void **setDirection**(*TransformDirection* dir) noexcept override

Note that this only affects the evaluation and not the values stored in the object.

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual void **validate**() const override

Will throw if data is not valid.

const char ***getSrc**() const

void **setSrc**(const char *src)

const char ***getDst**() const

void **setDst**(const char *dst)

bool **getDataBypass**() const noexcept

Data color spaces do not get processed when true (which is the default).

void **setDataBypass**(bool enabled) noexcept

ColorSpaceTransform &**operator=**(const *ColorSpaceTransform*&) = delete

virtual **~ColorSpaceTransform**()

Do not use (needed only for pybind11).

Public Static Functions

static *ColorSpaceTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ColorSpaceTransform*&)

typedef std::shared_ptr<const *ColorSpaceTransform*> OpenColorIO_v2_5::**ConstColorSpaceTransformRcPtr**

typedef std::shared_ptr<*ColorSpaceTransform*> OpenColorIO_v2_5::**ColorSpaceTransformRcPtr**

DisplayViewTransform

Python

class PyOpenColorIO.**DisplayViewTransform**

DisplayViewTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.DisplayViewTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.DisplayViewTransform, src: str = '', display: str = '', view: str = '', looksBypass: bool = False, dataBypass: bool = True, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

getDataBypass() → bool

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getDisplay() → str

getLooksBypass() → bool

getSrc() → str

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getView() → str

setDataBypass(*dataBypass: bool*) → None

Data color spaces do not get processed when true (which is the default).

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setDisplay(*display: str*) → None

Specify which display to use.

setLooksBypass(*looksBypass: bool*) → None

Looks will be bypassed when true (the default is false).

setSrc(*src: str*) → None

Specify the incoming color space.

setView(*view: str*) → None

Specify which view transform to use.

validate() → None

Will throw if data is not valid.

C++

class **DisplayViewTransform** : public *Transform*

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const override

virtual *TransformDirection* **getDirection**() const noexcept override

virtual void **setDirection**(*TransformDirection* dir) noexcept override

Note that this only affects the evaluation and not the values stored in the object.

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual void **validate**() const override

Will throw if data is not valid.

const char ***getSrc**() const

void **setSrc**(const char *name)

Specify the incoming color space.

const char *getDisplay() const

void setDisplay(const char *display)

Specify which display to use.

const char *getView() const

void setView(const char *view)

Specify which view transform to use.

bool getLooksBypass() const

void setLooksBypass(bool bypass)

Looks will be bypassed when true (the default is false).

bool getDataBypass() const noexcept

void setDataBypass(bool bypass) noexcept

Data color spaces do not get processed when true (which is the default).

virtual ~DisplayViewTransform()

Do not use (needed only for pybind11).

Public Static Functions

static *DisplayViewTransformRcPtr* Create()

std::ostream &OpenColorIO_v2_5::operator<<(std::ostream&, const *DisplayViewTransform*&)

typedef std::shared_ptr<const *DisplayViewTransform*>
OpenColorIO_v2_5::ConstDisplayViewTransformRcPtr

typedef std::shared_ptr<*DisplayViewTransform*> OpenColorIO_v2_5::DisplayViewTransformRcPtr

ExponentTransform

Python

class PyOpenColorIO.ExponentTransform

Represents exponent transform: pow(clamp(color), value).

Note: For configs with version == 1: Negative style is ignored and if the exponent is 1.0, this will not clamp. Otherwise, the input color will be clamped between [0.0, inf]. For configs with version > 1: Negative value handling may be specified via setNegativeStyle.

ExponentTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ExponentTransform) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.ExponentTransform, value: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(4)"] = [1.0, 1.0, 1.0, 1.0], negativeStyle: PyOpenColorIO.PyOpenColorIO.NegativeStyle = <NegativeStyle.NEGATIVE_CLAMP: 0>, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.ExponentTransform) → bool

Checks if this exactly equals other.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getNegativeStyle() → *PyOpenColorIO.PyOpenColorIO.NegativeStyle*

Specifies how negative values are handled. Legal values:

- NEGATIVE_CLAMP – Clamp negative values (default).
- NEGATIVE_MIRROR – Positive curve is rotated 180 degrees around the origin to handle negatives.
- NEGATIVE_PASS_THRU – Negative values are passed through unchanged.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue() → Annotated[list[float], 'FixedSize(4)']

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setNegativeStyle(*style:* PyOpenColorIO.PyOpenColorIO.NegativeStyle) → None

setValue(*value: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)']*) → None

validate() → None

Will throw if data is not valid.

C++

class **ExponentTransform** : public *Transform*

Represents exponent transform: pow(clamp(color), value).

Note: For configs with version == 1: Negative style is ignored and if the exponent is 1.0, this will not clamp. Otherwise, the input color will be clamped between [0.0, inf]. For configs with version > 1: Negative value handling may be specified via setNegativeStyle.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *ExponentTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual void **getValue**(double (&vec4)[4]) const noexcept = 0

virtual void **setValue**(const double (&vec4)[4]) noexcept = 0

virtual *NegativeStyle* **getNegativeStyle**() const = 0

Specifies how negative values are handled. Legal values:

- NEGATIVE_CLAMP — Clamp negative values (default).
- NEGATIVE_MIRROR — Positive curve is rotated 180 degrees around the origin to handle negatives.
- NEGATIVE_PASS_THRU — Negative values are passed through unchanged.

virtual void **setNegativeStyle**(*NegativeStyle* style) = 0

ExponentTransform(const *ExponentTransform*&) = delete

ExponentTransform &**operator=**(const *ExponentTransform*&) = delete

virtual **~ExponentTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *ExponentTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ExponentTransform*&)

typedef std::shared_ptr<const *ExponentTransform*> OpenColorIO_v2_5::**ConstExponentTransformRcPtr**

typedef std::shared_ptr<*ExponentTransform*> OpenColorIO_v2_5::**ExponentTransformRcPtr**

ExponentWithLinearTransform

Python

class PyOpenColorIO.**ExponentWithLinearTransform**

Represents power functions with a linear section in the shadows such as sRGB and L*.

The basic formula is:

pow((x + offset)/(1 + offset), gamma) with the breakpoint at offset/(gamma - 1).

Negative values are never clamped.

ExponentWithLinearTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ExponentWithLinearTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.ExponentWithLinearTransform, gamma: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(4)"] = [1.0, 1.0, 1.0, 1.0], offset: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat],

"FixedSize(4)"] = [0.0, 0.0, 0.0, 0.0], negativeStyle: PyOpenColorIO.PyOpenColorIO.NegativeStyle = <NegativeStyle.NEGATIVE_LINEAR: 3>, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.ExponentWithLinearTransform) → bool

Checks if this exactly equals other.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getGamma() → Annotated[list[float], 'FixedSize(4)']

getNegativeStyle() → *PyOpenColorIO.PyOpenColorIO.NegativeStyle*

Specifies how negative values are handled. Legal values:

- NEGATIVE_LINEAR – Linear segment continues into negatives (default).
- NEGATIVE_MIRROR – Positive curve is rotated 180 degrees around the origin to handle negatives.

getOffset() → Annotated[list[float], 'FixedSize(4)']

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setGamma(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)']*) → None

Set the exponent value for the power function for R, G, B, A.

Note: The gamma values must be in the range of [1, 10]. Set the transform direction to inverse to obtain the effect of values less than 1.

---

setNegativeStyle(*style:* PyOpenColorIO.PyOpenColorIO.NegativeStyle) → None

setOffset(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)']*) → None

Set the offset value for the power function for R, G, B, A.

Note: The offset values must be in the range [0, 0.9].

---

validate() → None

Will throw if data is not valid.

C++

class **ExponentWithLinearTransform** : public *Transform*

Represents power functions with a linear section in the shadows such as sRGB and L*.

The basic formula is::

pow((x + offset)/(1 + offset), gamma) with the breakpoint at offset/(gamma - 1).

Negative values are never clamped.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *ExponentWithLinearTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual void **getGamma**(double (&values)[4]) const noexcept = 0

virtual void **setGamma**(const double (&values)[4]) noexcept = 0

Set the exponent value for the power function for R, G, B, A.

Note: The gamma values must be in the range of [1, 10]. Set the transform direction to inverse to obtain the effect of values less than 1.

---

virtual void **getOffset**(double (&values)[4]) const noexcept = 0

virtual void **setOffset**(const double (&values)[4]) noexcept = 0

Set the offset value for the power function for R, G, B, A.

Note: The offset values must be in the range [0, 0.9].

---

virtual *NegativeStyle* **getNegativeStyle**() const = 0

Specifies how negative values are handled. Legal values:

- NEGATIVE_LINEAR — Linear segment continues into negatives (default).
- NEGATIVE_MIRROR — Positive curve is rotated 180 degrees around the origin to handle negatives.

virtual void **setNegativeStyle**(*NegativeStyle* style) = 0

ExponentWithLinearTransform(const *ExponentWithLinearTransform*&) = delete

ExponentWithLinearTransform &**operator=**(const *ExponentWithLinearTransform*&) = delete

virtual **~ExponentWithLinearTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *ExponentWithLinearTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ExponentWithLinearTransform*&)

typedef std::shared_ptr<const *ExponentWithLinearTransform*>
OpenColorIO_v2_5::**ConstExponentWithLinearTransformRcPtr**

typedef std::shared_ptr<*ExponentWithLinearTransform*>
OpenColorIO_v2_5::**ExponentWithLinearTransformRcPtr**

ExposureContrastTransform

Python

class PyOpenColorIO.**ExposureContrastTransform**

Applies exposure, gamma, and pivoted contrast adjustments. Adjusts the math to be appropriate for linear, logarithmic, or video color spaces.

ExposureContrastTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ExposureContrastTransform) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.ExposureContrastTransform, style: PyOpenColorIO.PyOpenColorIO.ExposureContrastStyle = <ExposureContrastStyle.EXPOSURE_CONTRAST_LINEAR: 0>, exposure: typing.SupportsFloat = 0.0, contrast: typing.SupportsFloat = 1.0, gamma: typing.SupportsFloat = 1.0, pivot: typing.SupportsFloat = 0.18, logExposureStep: typing.SupportsFloat = 0.088, logMidGray: typing.SupportsFloat = 0.435, dynamicExposure: bool = False, dynamicContrast: bool = False, dynamicGamma: bool = False, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.ExposureContrastTransform) → bool

Checks if this exactly equals other.

getContrast() → float

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getExposure() → float

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getGamma() → float

getLogExposureStep() → float

getLogMidGray() → float

getPivot() → float

getStyle() → *PyOpenColorIO.PyOpenColorIO.ExposureContrastStyle*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

isContrastDynamic() → bool

Contrast can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic contrast.

isExposureDynamic() → bool

Exposure can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic exposure.

isGammaDynamic() → bool

Gamma can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic gamma.

makeContrastDynamic() → None

makeContrastNonDynamic() → None

makeExposureDynamic() → None

makeExposureNonDynamic() → None

makeGammaDynamic() → None

makeGammaNonDynamic() → None

setContrast(*contrast: SupportsFloat*) → None

Applies a contrast/gamma adjustment around a pivot point. The contrast and gamma are mathematically the same, but two controls are provided to enable the use of separate dynamic parameters. Contrast is usually a scene-referred adjustment that pivots around gray whereas gamma is usually a display-referred adjustment that pivots around white.

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setExposure(*exposure: SupportsFloat*) → None

Applies an exposure adjustment. The value is in units of stops (regardless of style), for example, a value of -1 would be equivalent to reducing the lighting by one half.

setGamma(*gamma: SupportsFloat*) → None

setLogExposureStep(*logExposureStep: SupportsFloat*) → None

Set the increment needed to move one stop for the log-style algorithm. For example, ACEScct is 0.057, LogC is roughly 0.074, and Cineon is roughly 90/1023 = 0.088. The default value is 0.088.

setLogMidGray(*logMidGray: SupportsFloat*) → None

Set the position of 18% gray for use by the log-style algorithm. For example, ACEScct is about 0.41, LogC is about 0.39, and ADX10 is 445/1023 = 0.435. The default value is 0.435.

setPivot(*pivot: SupportsFloat*) → None

Set the pivot point around which the contrast and gamma controls will work. Regardless of whether linear/video/log-style is being used, the pivot is always expressed in linear. In other words, a pivot of 0.18 is always mid-gray.

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.ExposureContrastStyle) → None

Select the algorithm for linear, video or log color spaces.

validate() → None

Will throw if data is not valid.

C++

class **ExposureContrastTransform** : public *Transform*

Applies exposure, gamma, and pivoted contrast adjustments. Adjusts the math to be appropriate for linear, logarithmic, or video color spaces.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *ExposureContrastTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual *ExposureContrastStyle* **getStyle**() const = 0

virtual void **setStyle**(*ExposureContrastStyle* style) = 0

Select the algorithm for linear, video or log color spaces.

virtual double **getExposure**() const = 0

virtual void **setExposure**(double exposure) = 0

Applies an exposure adjustment. The value is in units of stops (regardless of style), for example, a value of -1 would be equivalent to reducing the lighting by one half.

virtual bool **isExposureDynamic**() const = 0

Exposure can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic exposure.

virtual void **makeExposureDynamic**() = 0

virtual void **makeExposureNonDynamic**() = 0

virtual double **getContrast**() const = 0

virtual void **setContrast**(double contrast) = 0

Applies a contrast/gamma adjustment around a pivot point. The contrast and gamma are mathematically the same, but two controls are provided to enable the use of separate dynamic parameters. Contrast is usually a scene-referred adjustment that pivots around gray whereas gamma is usually a display-referred adjustment that pivots around white.

virtual bool **isContrastDynamic**() const = 0

Contrast can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic contrast.

virtual void **makeContrastDynamic**() = 0

virtual void **makeContrastNonDynamic**() = 0

virtual double **getGamma**() const = 0

virtual void **setGamma**(double gamma) = 0

virtual bool **isGammaDynamic**() const = 0

Gamma can be made dynamic so the value can be changed through the CPU or GPU processor, but if there are several ExposureContrastTransform only one can have a dynamic gamma.

virtual void **makeGammaDynamic**() = 0

virtual void **makeGammaNonDynamic**() = 0

virtual double **getPivot**() const = 0

virtual void **setPivot**(double pivot) = 0

Set the pivot point around which the contrast and gamma controls will work. Regardless of whether linear/video/log-style is being used, the pivot is always expressed in linear. In other words, a pivot of 0.18 is always mid-gray.

virtual double **getLogExposureStep**() const = 0

virtual void **setLogExposureStep**(double logExposureStep) = 0

Set the increment needed to move one stop for the log-style algorithm. For example, ACEScct is 0.057, LogC is roughly 0.074, and Cineon is roughly 90/1023 = 0.088. The default value is 0.088.

virtual double **getLogMidGray**() const = 0

virtual void **setLogMidGray**(double logMidGray) = 0

Set the position of 18% gray for use by the log-style algorithm. For example, ACEScct is about 0.41, LogC is about 0.39, and ADX10 is 445/1023 = 0.435. The default value is 0.435.

virtual **~ExposureContrastTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *ExposureContrastTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ExposureContrastTransform*&)

typedef std::shared_ptr<const *ExposureContrastTransform*>
OpenColorIO_v2_5::**ConstExposureContrastTransformRcPtr**

typedef std::shared_ptr<*ExposureContrastTransform*>
OpenColorIO_v2_5::**ExposureContrastTransformRcPtr**

FileTransform

Python

class PyOpenColorIO.**FileTransform**

static IsFormatExtensionSupported(*extension: str*) → bool

Returns true if the extension corresponds to a format supported by FileTransform. The argument is case-insensitive, and a leading dot, if present, is ignored. Note that FileTransform will attempt all format readers on a given file until it is successful, even files that contain an unsupported extension or no extension. However, this function is useful for applications that want to know which files are likely to be LUT files, based on their extension.

FileTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.FileTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.FileTransform, src: str = '', cccId: str = '', interpolation: PyOpenColorIO.PyOpenColorIO.Interpolation = <Interpolation.INTERP_DEFAULT: 254>, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

getCCCId() → str

The cccid can be the ID of a CDL or the index of the CDL (as string). If cccid is NULL or empty the first CDL is returned. The cccid is case-sensitive.

getCDLStyle() → *PyOpenColorIO.PyOpenColorIO.CDLStyle*

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

static getFormats() → *PyOpenColorIO.PyOpenColorIO.FileTransform.FormatIterator*

getInterpolation() → *PyOpenColorIO.PyOpenColorIO.Interpolation*

The file parsers that care about interpolation (LUTs) will try to make use of the requested interpolation method when loading the file. In these cases, if the requested method could not be used, a warning is logged. If no method is provided, or a method cannot be used, INTERP_DEFAULT is used.

getSrc() → str

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setCCCId(*cccId: str*) → None

setCDLStyle(*style:* PyOpenColorIO.PyOpenColorIO.CDLStyle) → None

Can be used with CDL, CC & CCC formats to specify the clamping behavior of the CDLTransform. Default is CDL_NO_CLAMP.

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setInterpolation(*interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation) → None

setSrc(*src: str*) → None

validate() → None

Will throw if data is not valid.

class PyOpenColorIO.FileTransform.**FormatIterator**

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.FileTransform.FormatIterator*

len(*self*) → int

next(*self*) → tuple

C++

class **FileTransform** : public *Transform*

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const override

virtual *TransformDirection* **getDirection**() const noexcept override

virtual void **setDirection**(*TransformDirection* dir) noexcept override

Note that this only affects the evaluation and not the values stored in the object.

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual void **validate**() const override

Will throw if data is not valid.

const char ***getSrc**() const

void **setSrc**(const char *src)

const char ***getCCCId**() const

The cccid can be the ID of a CDL or the index of the CDL (as string). If cccid is NULL or empty the first CDL is returned. The cccid is case-sensitive.

void **setCCCId**(const char *id)

CDLStyle **getCDLStyle**() const

void **setCDLStyle**(*CDLStyle*)

Can be used with CDL, CC & CCC formats to specify the clamping behavior of the CDLTransform. Default is CDL_NO_CLAMP.

Interpolation **getInterpolation**() const

The file parsers that care about interpolation (LUTs) will try to make use of the requested interpolation method when loading the file. In these cases, if the requested method could not be used, a warning is logged. If no method is provided, or a method cannot be used, INTERP_DEFAULT is used.

void **setInterpolation**(*Interpolation* interp)

FileTransform &**operator=**(const *FileTransform*&) = delete

virtual **~FileTransform**()

Do not use (needed only for pybind11).

Public Static Functions

static *FileTransformRcPtr* **Create**()

static int **GetNumFormats**()

Get the number of LUT readers.

static const char ***GetFormatNameByIndex**(int index)

Get the LUT readers at index, return empty string if an invalid index is specified.

static const char ***GetFormatExtensionByIndex**(int index)

Get the LUT reader extension at index, return empty string if an invalid index is specified.

static bool **IsFormatExtensionSupported**(const char *extension)

Returns true if the extension corresponds to a format supported by FileTransform. The argument is case-insensitive, and a leading dot, if present, is ignored. Note that FileTransform will attempt all format readers on a given file until it is successful, even files that contain an unsupported extension or no extension. However, this function is useful for applications that want to know which files are likely to be LUT files, based on their extension.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *FileTransform*&)

typedef std::shared_ptr<const *FileTransform*> OpenColorIO_v2_5::**ConstFileTransformRcPtr**

typedef std::shared_ptr<*FileTransform*> OpenColorIO_v2_5::**FileTransformRcPtr**

FixedFunctionTransform

Python

class PyOpenColorIO.**FixedFunctionTransform**

Provides a set of hard-coded algorithmic building blocks that are needed to accurately implement various common color transformations.

FixedFunctionTransform(*style: PyOpenColorIO.PyOpenColorIO.FixedFunctionStyle*, *params: collections.abc.Sequence[typing.SupportsFloat] = []*, *direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>*) → None

equals(*other:* PyOpenColorIO.PyOpenColorIO.FixedFunctionTransform) → bool

Checks if this exactly equals other.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getParams() → list[float]

getStyle() → *PyOpenColorIO.PyOpenColorIO.FixedFunctionStyle*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setParams(*params: collections.abc.Sequence[SupportsFloat]*) → None

Set the parameters (for functions that require them).

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.FixedFunctionStyle) → None

Select which algorithm to use.

validate() → None

Will throw if data is not valid.

C++

class **FixedFunctionTransform** : public *Transform*

Provides a set of hard-coded algorithmic building blocks that are needed to accurately implement various common color transformations.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *FixedFunctionTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual *FixedFunctionStyle* **getStyle**() const = 0

virtual void **setStyle**(*FixedFunctionStyle* style) = 0

Select which algorithm to use.

virtual size_t **getNumParams**() const = 0

virtual void **getParams**(double *params) const = 0

virtual void **setParams**(const double *params, size_t num) = 0

Set the parameters (for functions that require them).

FixedFunctionTransform(const *FixedFunctionTransform*&) = delete

FixedFunctionTransform &**operator=**(const *FixedFunctionTransform*&) = delete

virtual **~FixedFunctionTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *FixedFunctionTransformRcPtr* **Create**(*FixedFunctionStyle* style)

static *FixedFunctionTransformRcPtr* **Create**(*FixedFunctionStyle* style, const double *params, size_t num)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *FixedFunctionTransform*&)

typedef std::shared_ptr<const *FixedFunctionTransform*>
OpenColorIO_v2_5::**ConstFixedFunctionTransformRcPtr**

typedef std::shared_ptr<*FixedFunctionTransform*> OpenColorIO_v2_5::**FixedFunctionTransformRcPtr**

GroupTransform

Python

class PyOpenColorIO.**GroupTransform**

static GetWriteFormats() → *PyOpenColorIO.PyOpenColorIO.GroupTransform.WriteFormatIterator*

GroupTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.GroupTransform) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.GroupTransform, transforms: collections.abc.Sequence[PyOpenColorIO.PyOpenColorIO.Transform] = [], direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

appendTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform) → None

Adds a transform to the end of the group.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

prependTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform) → None

Add a transform at the beginning of the group.

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

validate() → None

Will throw if data is not valid.

write(**args*, ***kwargs*)

Overloaded function.

1. write(self: PyOpenColorIO.PyOpenColorIO.GroupTransform, formatName: str, fileName: str, config: PyOpenColorIO.PyOpenColorIO.Config = None) -> None

Write the transforms comprising the group to the stream.

Writing (as opposed to Baking) is a lossless process. An exception is thrown if the processor cannot be losslessly written to the specified file format. Transforms such as FileTransform or ColorSpaceTransform are resolved into write-able simple transforms using the config and context. Supported formats include CTF, CLF, and CDL. All available formats can be listed with the following: .. code-block:: cpp

> // What are the allowed writing output formats? std::ostringstream formats; formats << "Formats to write to: "; for (int i = 0; i < :ref:`GroupTransform::GetNumWriteFormats`(); ++i) {
>
> > if (i != 0) formats << ", "; formats << :ref:`GroupTransform::GetFormatNameByIndex`(i); formats << " (." << GroupTransform::GetFormatExtensionByIndex(i) << ")";
>
> }

2. write(self: PyOpenColorIO.PyOpenColorIO.GroupTransform, formatName: str, config: PyOpenColorIO.PyOpenColorIO.Config = None) -> str

Write the transforms comprising the group to the stream.

Writing (as opposed to Baking) is a lossless process. An exception is thrown if the processor cannot be losslessly written to the specified file format. Transforms such as FileTransform or ColorSpaceTransform are resolved into write-able simple transforms using the config and context. Supported formats include CTF, CLF, and CDL. All available formats can be listed with the following: .. code-block:: cpp

> // What are the allowed writing output formats? std::ostringstream formats; formats << "Formats to write to: "; for (int i = 0; i < :ref:`GroupTransform::GetNumWriteFormats`(); ++i) {

```
            if (i != 0) formats << ", "; formats << :ref:`GroupTransform::GetFormatNameByIndex`(i);
            formats << " (." << GroupTransform::GetFormatExtensionByIndex(i) << ")";
        }
```

class PyOpenColorIO.GroupTransform.**WriteFormatIterator**

self [*arg0: SupportsInt*] → tuple

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GroupTransform.WriteFormatIterator*

len(*self*) → int

next(*self*) → tuple

class PyOpenColorIO.GroupTransform.**TransformIterator**

self [*arg0: SupportsInt*] → *PyOpenColorIO.PyOpenColorIO.Transform*

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.GroupTransform.TransformIterator*

len(*self*) → int

next(*self*) → *PyOpenColorIO.PyOpenColorIO.Transform*

C++

class **GroupTransform** : public *Transform*

Public Functions

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual *ConstTransformRcPtr* **getTransform**(int index) const = 0

Throws if index is not allowed.

virtual *TransformRcPtr* &**getTransform**(int index) = 0

Throws if index is not allowed.

virtual int **getNumTransforms**() const noexcept = 0

Return number of transforms.

virtual void **appendTransform**(*TransformRcPtr* transform) noexcept = 0

Adds a transform to the end of the group.

virtual void **prependTransform**(*TransformRcPtr* transform) noexcept = 0

Add a transform at the beginning of the group.

virtual void **write**(const *ConstConfigRcPtr* &config, const char *formatName, std::ostream &os) const = 0

Write the transforms comprising the group to the stream.

Writing (as opposed to Baking) is a lossless process. An exception is thrown if the processor cannot be losslessly written to the specified file format. Transforms such as FileTransform or ColorSpaceTransform are resolved into write-able simple transforms using the config and context. Supported formats include CTF, CLF, and CDL. All available formats can be listed with the following:

```
// What are the allowed writing output formats?
std::ostringstream formats;
formats << "Formats to write to: ";
for (int i = 0; i < GroupTransform::GetNumWriteFormats(); ++i)
{
   if (i != 0) formats << ", ";
   formats << GroupTransform::GetFormatNameByIndex(i);
   formats << " (." << GroupTransform::GetFormatExtensionByIndex(i) << ")";
}
```

GroupTransform(const *GroupTransform*&) = delete

GroupTransform &**operator=**(const *GroupTransform*&) = delete

virtual **~GroupTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *GroupTransformRcPtr* **Create**()

static int **GetNumWriteFormats**() noexcept

Get the number of writers.

static const char ***GetFormatNameByIndex**(int index) noexcept

Get the writer at index, return empty string if an invalid index is specified.

static const char ***GetFormatExtensionByIndex**(int index) noexcept

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *GroupTransform*&)

typedef std::shared_ptr<const *GroupTransform*> OpenColorIO_v2_5::**ConstGroupTransformRcPtr**

typedef std::shared_ptr<*GroupTransform*> OpenColorIO_v2_5::**GroupTransformRcPtr**

LogAffineTransform

Python

class PyOpenColorIO.**LogAffineTransform**

Applies a logarithm with an affine transform before and after. Represents the Cineon lin-to-log type transforms:

logSideSlope * log(linSideSlope * color + linSideOffset, base) + logSideOffset

- Default values are: 1. * log(1. * color + 0., 2.) + 0.
- The alpha channel is not affected.

LogAffineTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.LogAffineTransform) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.LogAffineTransform, logSideSlope: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [1.0, 1.0, 1.0], logSideOffset: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [0.0, 0.0, 0.0], linSideSlope: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [1.0, 1.0, 1.0], linSideOffset: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(3)"] = [0.0, 0.0, 0.0], direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.LogAffineTransform) → bool

Checks if this exactly equals other.

getBase() → float

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getLinSideOffsetValue() → Annotated[list[float], 'FixedSize(3)']

getLinSideSlopeValue() → Annotated[list[float], 'FixedSize(3)']

getLogSideOffsetValue() → Annotated[list[float], 'FixedSize(3)']

getLogSideSlopeValue() → Annotated[list[float], 'FixedSize(3)']

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setBase(*base: SupportsFloat*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setLinSideOffsetValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLinSideSlopeValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLogSideOffsetValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLogSideSlopeValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

validate() → None

Will throw if data is not valid.

C++

class **LogAffineTransform** : public *Transform*

Applies a logarithm with an affine transform before and after. Represents the Cineon lin-to-log type transforms::

logSideSlope * log(linSideSlope * color + linSideOffset, base) + logSideOffset

- Default values are: 1. * log(1. * color + 0., 2.) + 0.
- The alpha channel is not affected.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *LogAffineTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual double **getBase**() const noexcept = 0

virtual void **setBase**(double base) noexcept = 0

virtual void **getLogSideSlopeValue**(double (&values)[3]) const noexcept = 0

virtual void **setLogSideSlopeValue**(const double (&values)[3]) noexcept = 0

virtual void **getLogSideOffsetValue**(double (&values)[3]) const noexcept = 0

virtual void **setLogSideOffsetValue**(const double (&values)[3]) noexcept = 0

virtual void **getLinSideSlopeValue**(double (&values)[3]) const noexcept = 0

virtual void **setLinSideSlopeValue**(const double (&values)[3]) noexcept = 0

virtual void **getLinSideOffsetValue**(double (&values)[3]) const noexcept = 0

virtual void **setLinSideOffsetValue**(const double (&values)[3]) noexcept = 0

LogAffineTransform(const *LogAffineTransform*&) = delete

LogAffineTransform &**operator=**(const *LogAffineTransform*&) = delete

virtual **~LogAffineTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *LogAffineTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *LogAffineTransform*&)

typedef std::shared_ptr<const *LogAffineTransform*> OpenColorIO_v2_5::**ConstLogAffineTransformRcPtr**

typedef std::shared_ptr<*LogAffineTransform*> OpenColorIO_v2_5::**LogAffineTransformRcPtr**

LogCameraTransform

Python

class PyOpenColorIO.**LogCameraTransform**

Same as LogAffineTransform but with the addition of a linear segment near black. This formula is used for many camera logs (e.g., LogC) as well as ACEScct.

- The linSideBreak specifies the point on the linear axis where the log and linear segments meet. It must be set (there is no default).
- The linearSlope specifies the slope of the linear segment of the forward (linToLog) transform. By default it is set equal to the slope of the log curve at the break point.

LogCameraTransform → None

LinSideBreak must be set for the transform to be valid (there is no default).

equals(*other:* PyOpenColorIO.PyOpenColorIO.LogCameraTransform) → bool

Checks if this exactly equals other.

getBase() → float

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getLinSideBreakValue() → Annotated[list[float], 'FixedSize(3)']

getLinSideOffsetValue() → Annotated[list[float], 'FixedSize(3)']

getLinSideSlopeValue() → Annotated[list[float], 'FixedSize(3)']

getLinearSlopeValue() → Annotated[list[float], 'FixedSize(3)']

Return LinearSlope or 3 qnan values if not defined.

getLogSideOffsetValue() → Annotated[list[float], 'FixedSize(3)']

getLogSideSlopeValue() → Annotated[list[float], 'FixedSize(3)']

Get/Set values for the R, G, B components.

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

isLinearSlopeValueSet() → bool

setBase(*base: SupportsFloat*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setLinSideBreakValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLinSideOffsetValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLinSideSlopeValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLinearSlopeValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

Set LinearSlope value.

Note: You must call setLinSideBreakValue before calling this.

---

setLogSideOffsetValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

setLogSideSlopeValue(*values: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → None

unsetLinearSlopeValue() → None

Remove LinearSlope values so that default values are used.

validate() → None

Will throw if data is not valid.

C++

class **LogCameraTransform** : public *Transform*

Same as LogAffineTransform but with the addition of a linear segment near black. This formula is used for many camera logs (e.g., LogC) as well as ACEScct.

- The linSideBreak specifies the point on the linear axis where the log and linear segments meet. It must be set (there is no default).
- The linearSlope specifies the slope of the linear segment of the forward (linToLog) transform. By default it is set equal to the slope of the log curve at the break point.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *LogCameraTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual double **getBase**() const noexcept = 0

virtual void **setBase**(double base) noexcept = 0

virtual void **getLogSideSlopeValue**(double (&values)[3]) const noexcept = 0

Get/Set values for the R, G, B components.

virtual void **setLogSideSlopeValue**(const double (&values)[3]) noexcept = 0

virtual void **getLogSideOffsetValue**(double (&values)[3]) const noexcept = 0

virtual void **setLogSideOffsetValue**(const double (&values)[3]) noexcept = 0

virtual void **getLinSideSlopeValue**(double (&values)[3]) const noexcept = 0

virtual void **setLinSideSlopeValue**(const double (&values)[3]) noexcept = 0

virtual void **getLinSideOffsetValue**(double (&values)[3]) const noexcept = 0

virtual void **setLinSideOffsetValue**(const double (&values)[3]) noexcept = 0

virtual void **getLinSideBreakValue**(double (&values)[3]) const noexcept = 0

virtual void **setLinSideBreakValue**(const double (&values)[3]) noexcept = 0

virtual bool **getLinearSlopeValue**(double (&values)[3]) const = 0
Return true if LinearSlope values were set, false if they were not.

virtual void **setLinearSlopeValue**(const double (&values)[3]) = 0
Set LinearSlope value.

Note: You must call setLinSideBreakValue before calling this.

---

virtual void **unsetLinearSlopeValue**() = 0
Remove LinearSlope values so that default values are used.

LogCameraTransform(const *LogCameraTransform*&) = delete

LogCameraTransform &**operator=**(const *LogCameraTransform*&) = delete

virtual **~LogCameraTransform**() = default
Do not use (needed only for pybind11).

Public Static Functions

static *LogCameraTransformRcPtr* **Create**(const double (&linSideBreakValues)[3])
LinSideBreak must be set for the transform to be valid (there is no default).

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *LogCameraTransform*&)

typedef std::shared_ptr<const *LogCameraTransform*> OpenColorIO_v2_5::**ConstLogCameraTransformRcPtr**

typedef std::shared_ptr<*LogCameraTransform*> OpenColorIO_v2_5::**LogCameraTransformRcPtr**

LogTransform

Python

class PyOpenColorIO.**LogTransform**
Represents log transform: log(color, base)

- The input will be clamped for negative numbers.
- Default base is 2.0.
- The alpha channel is not affected.

LogTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.LogTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.LogTransform, base: typing.SupportsFloat = 2.0, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.LogTransform) → bool

Checks if this exactly equals other.

getBase() → float

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setBase(*base: SupportsFloat*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

validate() → None

Will throw if data is not valid.

C++

class **LogTransform** : public *Transform*

Represents log transform: log(color, base)

- The input will be clamped for negative numbers.
- Default base is 2.0.
- The alpha channel is not affected.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *LogTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual double **getBase**() const noexcept = 0

virtual void **setBase**(double val) noexcept = 0

LogTransform(const *LogTransform*&) = delete

LogTransform &**operator=**(const *LogTransform*&) = delete

virtual **~LogTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *LogTransformRcPtr* **Create**()

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *LogTransform*&)

typedef std::shared_ptr<const *LogTransform*> OpenColorIO_v2_5::**ConstLogTransformRcPtr**

typedef std::shared_ptr<*LogTransform*> OpenColorIO_v2_5::**LogTransformRcPtr**

LookTransform

Python

class PyOpenColorIO.**LookTransform**

LookTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.LookTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.LookTransform, src: str, dst: str, looks: str = '', skipColorSpaceConversion: bool = False, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getDst() → str

getLooks() → str

getSkipColorSpaceConversion() → bool

getSrc() → str

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setDst(*dst: str*) → None

setLooks(*looks: str*) → None

Specify looks to apply. Looks is a potentially comma (or colon) delimited list of look names, Where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either)

setSkipColorSpaceConversion(*skipColorSpaceConversion: bool*) → None

setSrc(*src: str*) → None

validate() → None

Will throw if data is not valid.

C++

class **LookTransform** : public *Transform*

Public Functions

virtual *TransformRcPtr* **createEditableCopy**() const override

virtual *TransformDirection* **getDirection**() const noexcept override

virtual void **setDirection**(*TransformDirection* dir) noexcept override

Note that this only affects the evaluation and not the values stored in the object.

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual void **validate**() const override

Will throw if data is not valid.

const char ***getSrc**() const

void **setSrc**(const char *src)

const char ***getDst**() const

void **setDst**(const char *dst)

const char ***getLooks**() const

void **setLooks**(const char *looks)

Specify looks to apply. Looks is a potentially comma (or colon) delimited list of look names, Where +/- prefixes are optionally allowed to denote forward/inverse look specification. (And forward is assumed in the absence of either)

bool **getSkipColorSpaceConversion**() const

void **setSkipColorSpaceConversion**(bool skip)

LookTransform &**operator=**(const *LookTransform*&) = delete

virtual **~LookTransform**()

Do not use (needed only for pybind11).

Public Static Functions

static *LookTransformRcPtr* **Create**()

static const char ***GetLooksResultColorSpace**(const *ConstConfigRcPtr* &config, const *ConstContextRcPtr* &context, const char *looks)

Return the name of the color space after applying looks in the forward direction but without converting to the destination color space. This is equivalent to the process space of the last look in the look sequence (and takes into account that a look fall-back may be used).

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *LookTransform*&)

typedef std::shared_ptr<const *LookTransform*> OpenColorIO_v2_5::**ConstLookTransformRcPtr**

typedef std::shared_ptr<*LookTransform*> OpenColorIO_v2_5::**LookTransformRcPtr**

Lut1DTransform

Python

class PyOpenColorIO.**Lut1DTransform**

Represents a 1D-LUT transform.

Lut1DTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut1DTransform) -> None

Create an identity 1D-LUT of length two.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut1DTransform, length: typing.SupportsInt, inputHalfDomain: bool) -> None

Create an identity 1D-LUT with specific length and half-domain setting. Will throw for lengths longer than 1024x1024.

3. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut1DTransform, length: typing.SupportsInt = 2, inputHalfDomain: bool = False, outputRawHalfs: bool = False, fileOutputBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth = <BitDepth.BIT_DEPTH_UNKNOWN: 0>, hueAdjust: PyOpenColorIO.PyOpenColorIO.Lut1DHueAdjust = <Lut1DHueAdjust.HUE_NONE: 0>, interpolation: PyOpenColorIO.PyOpenColorIO.Interpolation = <Interpolation.INTERP_DEFAULT: 254>, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Create an identity 1D-LUT with specific length and half-domain setting. Will throw for lengths longer than 1024x1024.

equals(*other:* PyOpenColorIO.PyOpenColorIO.Lut1DTransform) → bool

Checks if this exactly equals other.

getData() → numpy.ndarray

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFileOutputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getHueAdjust() → *PyOpenColorIO.PyOpenColorIO.Lut1DHueAdjust*

getInputHalfDomain() → bool

getInterpolation() → *PyOpenColorIO.PyOpenColorIO.Interpolation*

getLength() → int

getOutputRawHalfs() → bool

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue(*index: SupportsInt*) → tuple

setData(*data: typing_extensions.Buffer*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setFileOutputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

Get the bit-depth associated with the LUT values read from a file or set the bit-depth of values to be written to a file (for file formats such as CLF that support multiple bit-depths). However, note that the values stored in the object are always normalized.

setHueAdjust(*hueAdjust:* PyOpenColorIO.PyOpenColorIO.Lut1DHueAdjust) → None

The 1D-LUT transform optionally supports a hue adjustment feature that was used in some versions of ACES. This adjusts the hue of the result to approximately match the input.

setInputHalfDomain(*isHalfDomain: bool*) → None

In a half-domain LUT, the contents of the LUT specify the desired value of the function for each half-float value. Therefore, the length of the LUT must be 65536 entries or else validate() will throw.

setInterpolation(*interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation) → None

setLength(*length: SupportsInt*) → None

Changing the length will reset the LUT to identity. Will throw for lengths longer than 1024x1024.

setOutputRawHalfs(*isRawHalfs: bool*) → None

Set OutputRawHalfs to true if you want to output the LUT contents as 16-bit floating point values expressed as unsigned 16-bit integers representing the equivalent bit pattern. For example, the value 1.0 would be written as the integer 15360 because it has the same bit-pattern. Note that this implies the values will be quantized to a 16-bit float. Note that this setting only controls the output formatting (where supported) and not the values for getValue/setValue. The only file formats that currently support this are CLF and CTF.

setValue(*index: SupportsInt*, *r: SupportsFloat*, *g: SupportsFloat*, *b: SupportsFloat*) → None

Set the values of a LUT1D. Will throw if the index is outside of the range from 0 to (length-1).

The LUT values are always for the "forward" LUT, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in any given LUT files. For example in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

LUTs in various file formats may only provide values for one channel where R, G, B are the same. Even in that case, you should provide three equal values to the setter.

validate() → None

Will throw if data is not valid.

C++

class **Lut1DTransform** : public *Transform*

Represents a 1D-LUT transform.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual *BitDepth* **getFileOutputBitDepth**() const noexcept = 0

virtual void **setFileOutputBitDepth**(*BitDepth* bitDepth) noexcept = 0

Get the bit-depth associated with the LUT values read from a file or set the bit-depth of values to be written to a file (for file formats such as CLF that support multiple bit-depths). However, note that the values stored in the object are always normalized.

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *Lut1DTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual unsigned long **getLength**() const = 0

virtual void **setLength**(unsigned long length) = 0

Changing the length will reset the LUT to identity. Will throw for lengths longer than 1024x1024.

virtual void **getValue**(unsigned long index, float &r, float &g, float &b) const = 0

virtual void **setValue**(unsigned long index, float r, float g, float b) = 0

Set the values of a LUT1D. Will throw if the index is outside of the range from 0 to (length-1).

The LUT values are always for the "forward" LUT, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in any given LUT files. For example in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

LUTs in various file formats may only provide values for one channel where R, G, B are the same. Even in that case, you should provide three equal values to the setter.

virtual bool **getInputHalfDomain**() const noexcept = 0

virtual void **setInputHalfDomain**(bool isHalfDomain) noexcept = 0

In a half-domain LUT, the contents of the LUT specify the desired value of the function for each half-float value. Therefore, the length of the LUT must be 65536 entries or else validate() will throw.

virtual bool **getOutputRawHalfs**() const noexcept = 0

virtual void **setOutputRawHalfs**(bool isRawHalfs) noexcept = 0

Set OutputRawHalfs to true if you want to output the LUT contents as 16-bit floating point values expressed as unsigned 16-bit integers representing the equivalent bit pattern. For example, the value 1.0 would be written as the integer 15360 because it has the same bit-pattern. Note that this implies the values will be quantized to a 16-bit float. Note that this setting only controls the output formatting (where supported) and not the values for getValue/setValue. The only file formats that currently support this are CLF and CTF.

virtual *Lut1DHueAdjust* **getHueAdjust**() const noexcept = 0

virtual void **setHueAdjust**(*Lut1DHueAdjust* algo) = 0

The 1D-LUT transform optionally supports a hue adjustment feature that was used in some versions of ACES. This adjusts the hue of the result to approximately match the input.

virtual *Interpolation* **getInterpolation**() const = 0

virtual void **setInterpolation**(*Interpolation* algo) = 0

Lut1DTransform(const *Lut1DTransform*&) = delete

Lut1DTransform &**operator=**(const *Lut1DTransform*&) = delete

virtual **~Lut1DTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *Lut1DTransformRcPtr* **Create**()

Create an identity 1D-LUT of length two.

static *Lut1DTransformRcPtr* **Create**(unsigned long length, bool isHalfDomain)

Create an identity 1D-LUT with specific length and half-domain setting. Will throw for lengths longer than 1024x1024.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *Lut1DTransform*&)

typedef std::shared_ptr<const *Lut1DTransform*> OpenColorIO_v2_5::**ConstLut1DTransformRcPtr**

typedef std::shared_ptr<*Lut1DTransform*> OpenColorIO_v2_5::**Lut1DTransformRcPtr**

Lut3DTransform

Python

class PyOpenColorIO.**Lut3DTransform**

Represents a 3D-LUT transform.

Lut3DTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut3DTransform) -> None

Create an identity 3D-LUT of size 2x2x2.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut3DTransform, gridSize: typing.SupportsInt) -> None

Create an identity 3D-LUT with specific grid size. Will throw for grid size larger than 129.

3. __init__(self: PyOpenColorIO.PyOpenColorIO.Lut3DTransform, gridSize: typing.SupportsInt = 2, fileOutputBitDepth: PyOpenColorIO.PyOpenColorIO.BitDepth = <BitDepth.BIT_DEPTH_UNKNOWN: 0>, interpolation: PyOpenColorIO.PyOpenColorIO.Interpolation = <Interpolation.INTERP_DEFAULT: 254>, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Create an identity 3D-LUT with specific grid size. Will throw for grid size larger than 129.

equals(*other:* PyOpenColorIO.PyOpenColorIO.Lut3DTransform) → bool

Checks if this exactly equals other.

getData() → numpy.ndarray

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFileOutputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getGridSize() → int

getInterpolation() → *PyOpenColorIO.PyOpenColorIO.Interpolation*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

getValue(*indexR: SupportsInt*, *indexG: SupportsInt*, *indexB: SupportsInt*) → tuple

setData(*data: typing_extensions.Buffer*) → None

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setFileOutputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

Get the bit-depth associated with the LUT values read from a file or set the bit-depth of values to be written to a file (for file formats such as CLF that support multiple bit-depths). However, note that the values stored in the object are always normalized.

setGridSize(*gridSize: SupportsInt*) → None

Changing the grid size will reset the LUT to identity. Will throw for grid sizes larger than 129.

setInterpolation(*interpolation:* PyOpenColorIO.PyOpenColorIO.Interpolation) → None

setValue(*indexR: SupportsInt*, *indexG: SupportsInt*, *indexB: SupportsInt*, *r: SupportsFloat*, *g: SupportsFloat*, *b: SupportsFloat*) → None

Set the values of a 3D-LUT. Will throw if an index is outside of the range from 0 to (gridSize-1).

The LUT values are always for the "forward" LUT, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in any given LUT files. For example in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

validate() → None

Will throw if data is not valid.

C++

class **Lut3DTransform** : public *Transform*

Represents a 3D-LUT transform.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual *BitDepth* **getFileOutputBitDepth**() const noexcept = 0

virtual void **setFileOutputBitDepth**(*BitDepth* bitDepth) noexcept = 0

Get the bit-depth associated with the LUT values read from a file or set the bit-depth of values to be written to a file (for file formats such as CLF that support multiple bit-depths). However, note that the values stored in the object are always normalized.

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *Lut3DTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual unsigned long **getGridSize**() const = 0

virtual void **setGridSize**(unsigned long gridSize) = 0

Changing the grid size will reset the LUT to identity. Will throw for grid sizes larger than 129.

virtual void **getValue**(unsigned long indexR, unsigned long indexG, unsigned long indexB, float &r, float &g, float &b) const = 0

virtual void **setValue**(unsigned long indexR, unsigned long indexG, unsigned long indexB, float r, float g, float b) = 0

Set the values of a 3D-LUT. Will throw if an index is outside of the range from 0 to (gridSize-1).

The LUT values are always for the "forward" LUT, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in any given LUT files. For example in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

virtual *Interpolation* **getInterpolation**() const = 0

virtual void **setInterpolation**(*Interpolation* algo) = 0

Lut3DTransform(const *Lut3DTransform*&) = delete

Lut3DTransform &**operator=**(const *Lut3DTransform*&) = delete

virtual **~Lut3DTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *Lut3DTransformRcPtr* **Create**()

Create an identity 3D-LUT of size 2x2x2.

static *Lut3DTransformRcPtr* **Create**(unsigned long gridSize)

Create an identity 3D-LUT with specific grid size. Will throw for grid size larger than 129.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *Lut3DTransform*&)

typedef std::shared_ptr<const *Lut3DTransform*> OpenColorIO_v2_5::**ConstLut3DTransformRcPtr**

typedef std::shared_ptr<*Lut3DTransform*> OpenColorIO_v2_5::**Lut3DTransformRcPtr**

MatrixTransform

Python

class PyOpenColorIO.**MatrixTransform**

Represents an MX+B Matrix transform.

Note: For singular matrices, an inverse direction will throw an exception during finalization.

---

static Fit(*oldMin: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)'] = [0.0, 0.0, 0.0, 0.0]*, *oldMax: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)'] = [1.0, 1.0, 1.0, 1.0]*, *newMin: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)'] = [0.0, 0.0, 0.0, 0.0]*, *newMax: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)'] = [1.0, 1.0, 1.0, 1.0]*) → *PyOpenColorIO.PyOpenColorIO.MatrixTransform*

Convenience functions

Build the matrix and offset corresponding to higher-level concepts.

Note: These can throw an exception if for any component *oldmin == oldmax. (divide by 0)*

---

static Identity() → *PyOpenColorIO.PyOpenColorIO.MatrixTransform*

static Sat(*sat: SupportsFloat*, *lumaCoef: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → *PyOpenColorIO.PyOpenColorIO.MatrixTransform*

static Scale(*scale: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)']*) → *PyOpenColorIO.PyOpenColorIO.MatrixTransform*

static View(*channelHot: Annotated[collections.abc.Sequence[SupportsInt], 'FixedSize(4)']*, *lumaCoef: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(3)']*) → *PyOpenColorIO.PyOpenColorIO.MatrixTransform*

MatrixTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.MatrixTransform) -> None
2. __init__(self: PyOpenColorIO.PyOpenColorIO.MatrixTransform, matrix: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(16)"] = [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0], offset: typing.Annotated[collections.abc.Sequence[typing.SupportsFloat], "FixedSize(4)"] = [0.0, 0.0, 0.0, 0.0], direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

equals(*other:* PyOpenColorIO.PyOpenColorIO.MatrixTransform) → bool

Checks if this exactly equals other.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFileInputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

Get the bit-depths associated with the matrix values read from a file or set the bit-depths of values to be written to a file (for file formats such as CLF that support multiple bit-depths).

In a format such as CLF, the matrix values are scaled to take pixels at the specified inBitDepth to pixels at the specified outBitDepth. This complicates the interpretation of the matrix values and so this object always holds normalized values and scaling is done on the way from or to file formats such as CLF.

getFileOutputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getMatrix() → Annotated[list[float], 'FixedSize(16)']

getOffset() → Annotated[list[float], 'FixedSize(4)']

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setFileInputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

setFileOutputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

setMatrix(*matrix: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(16)']*) → None

Get or set the values of a Matrix. Expects 16 values, where the first four are the coefficients to generate the R output channel from R, G, B, A input channels.

The Matrix values are always for the "forward" Matrix, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in file formats such as CLF. For example in a CLF file using a "32f" input depth and "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

setOffset(*offset: Annotated[collections.abc.Sequence[SupportsFloat], 'FixedSize(4)']*) → None

Get or set the R, G, B, A offsets to be applied after the matrix.

These values are normalized relative to what may be stored in file formats such as CLF. For example, in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

validate() → None

Will throw if data is not valid.

C++

class **MatrixTransform** : public *Transform*

Represents an MX+B Matrix transform.

Note: For singular matrices, an inverse direction will throw an exception during finalization.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *MatrixTransform* &other) const noexcept = 0

Checks if this exactly equals other.

virtual void **getMatrix**(double *m44) const = 0

virtual void **setMatrix**(const double *m44) = 0

Get or set the values of a Matrix. Expects 16 values, where the first four are the coefficients to generate the R output channel from R, G, B, A input channels.

The Matrix values are always for the "forward" Matrix, regardless of how the transform direction is set.

These values are normalized relative to what may be stored in file formats such as CLF. For example in a CLF file using a "32f" input depth and "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

virtual void **getOffset**(double *offset4) const = 0

virtual void **setOffset**(const double *offset4) = 0

Get or set the R, G, B, A offsets to be applied after the matrix.

These values are normalized relative to what may be stored in file formats such as CLF. For example, in a CLF file using a "10i" output depth, a value of 1023 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

virtual *BitDepth* **getFileInputBitDepth**() const noexcept = 0

Get the bit-depths associated with the matrix values read from a file or set the bit-depths of values to be written to a file (for file formats such as CLF that support multiple bit-depths).

In a format such as CLF, the matrix values are scaled to take pixels at the specified inBitDepth to pixels at the specified outBitDepth. This complicates the interpretation of the matrix values and so this object always holds normalized values and scaling is done on the way from or to file formats such as CLF.

virtual void **setFileInputBitDepth**(*BitDepth* bitDepth) noexcept = 0

virtual *BitDepth* **getFileOutputBitDepth**() const noexcept = 0

virtual void **setFileOutputBitDepth**(*BitDepth* bitDepth) noexcept = 0

MatrixTransform(const *MatrixTransform*&) = delete

MatrixTransform &**operator=**(const *MatrixTransform*&) = delete

virtual **~MatrixTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *MatrixTransformRcPtr* **Create**()

static void **Fit**(double *m44, double *offset4, const double *oldmin4, const double *oldmax4, const double *newmin4, const double *newmax4)

Convenience functions

Build the matrix and offset corresponding to higher-level concepts.

Note: These can throw an exception if for any component `oldmin == oldmax. (divide by 0)`

static void **Identity**(double *m44, double *offset4)

static void **Sat**(double *m44, double *offset4, double sat, const double *lumaCoef3)

static void **Scale**(double *m44, double *offset4, const double *scale4)

static void **View**(double *m44, double *offset4, int *channelHot4, const double *lumaCoef3)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *MatrixTransform*&) noexcept

typedef std::shared_ptr<const *MatrixTransform*> OpenColorIO_v2_5::**ConstMatrixTransformRcPtr**

typedef std::shared_ptr<*MatrixTransform*> OpenColorIO_v2_5::**MatrixTransformRcPtr**

RangeTransform

Python

class PyOpenColorIO.**RangeTransform**

Represents a range transform

The Range is used to apply an affine transform (scale & offset) and clamps values to min/max bounds on all color components except the alpha. The scale and offset values are computed from the input and output bounds.

Refer to section 7.2.4 in specification S-2014-006 "A Common File Format for Look-Up Tables" from the Academy of Motion Picture Arts and Sciences and the American Society of Cinematographers.

Note: The "noClamp" style described in the specification S-2014-006 becomes a MatrixOp at the processor level.

Note: Changing the transform direction does not modify the in/out values – they are always specified with respect to the "forward" direction.

RangeTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.RangeTransform) -> None

Creates an instance of RangeTransform.

2. __init__(self: PyOpenColorIO.PyOpenColorIO.RangeTransform, minInValue: typing.SupportsFloat = nan, maxInValue: typing.SupportsFloat = nan, minOutValue: typing.SupportsFloat = nan, maxOutValue: typing.SupportsFloat = nan, direction: PyOpenColorIO.PyOpenColorIO.TransformDirection = <TransformDirection.TRANSFORM_DIR_FORWARD: 0>) -> None

Creates an instance of RangeTransform.

equals(*other:* PyOpenColorIO.PyOpenColorIO.RangeTransform) → bool

Checks if this equals other.

getDirection() → *PyOpenColorIO.PyOpenColorIO.TransformDirection*

getFileInputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

File bit-depth

In a format such as CLF, the range values are scaled to take pixels at the specified inBitDepth to pixels at the specified outBitDepth. This complicates the interpretation of the range values and so this object always

holds normalized values and scaling is done on the way from or to file formats such as CLF.Get the bit-depths associated with the range values read from a file or set the bit-depths of values to be written to a file (for file formats such as CLF that support multiple bit-depths).

getFileOutputBitDepth() → *PyOpenColorIO.PyOpenColorIO.BitDepth*

getFormatMetadata() → *PyOpenColorIO.PyOpenColorIO.FormatMetadata*

getMaxInValue() → float

Get the maximum value for the input.

getMaxOutValue() → float

Get the maximum value for the output.

getMinInValue() → float

Get the minimum value for the input.

Range values

These values are normalized relative to what may be stored in file formats such as CLF. For example in a CLF file using a "10i" input depth, a MaxInValue of 1023 in the file is normalized to 1.0. Likewise, for an output depth of "12i", a MaxOutValue of 4095 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

getMinOutValue() → float

Get the minimum value for the output.

getStyle() → *PyOpenColorIO.PyOpenColorIO.RangeStyle*

getTransformType() → *PyOpenColorIO.PyOpenColorIO.TransformType*

hasMaxInValue() → bool

Is the maximum value for the input set?

hasMaxOutValue() → bool

Is the maximum value for the output set?

hasMinInValue() → bool

Is the minimum value for the input set?

hasMinOutValue() → bool

Is the minimum value for the output set?

setDirection(*direction:* PyOpenColorIO.PyOpenColorIO.TransformDirection) → None

Note that this only affects the evaluation and not the values stored in the object.

setFileInputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

setFileOutputBitDepth(*bitDepth:* PyOpenColorIO.PyOpenColorIO.BitDepth) → None

setMaxInValue(*value: SupportsFloat*) → None

Set the maximum value for the input.

setMaxOutValue(*value: SupportsFloat*) → None

Set the maximum value for the output.

setMinInValue(*value: SupportsFloat*) → None

Set the minimum value for the input.

setMinOutValue(*value: SupportsFloat*) → None

Set the minimum value for the output.

setStyle(*style:* PyOpenColorIO.PyOpenColorIO.RangeStyle) → None

Set the Range style to clamp or not input values.

unsetMaxInValue() → None

Unset the maximum value for the input.

unsetMaxOutValue() → None

Unset the maximum value for the output.

unsetMinInValue() → None

Unset the minimum value for the input.

unsetMinOutValue() → None

Unset the minimum value for the output.

validate() → None

Will throw if data is not valid.

C++

class **RangeTransform** : public *Transform*

Represents a range transform

The Range is used to apply an affine transform (scale & offset) and clamps values to min/max bounds on all color components except the alpha. The scale and offset values are computed from the input and output bounds.

Refer to section 7.2.4 in specification S-2014-006 "A Common File Format

for Look-Up Tables" from the Academy of Motion Picture Arts and Sciences and the American Society of Cinematographers.

Note: The "noClamp" style described in the specification S-2014-006 becomes a MatrixOp at the processor level.

Note: Changing the transform direction does not modify the in/out values — they are always specified with respect to the "forward" direction.

Public Functions

inline virtual *TransformType* **getTransformType**() const noexcept override

virtual *RangeStyle* **getStyle**() const noexcept = 0

virtual void **setStyle**(*RangeStyle* style) noexcept = 0

Set the Range style to clamp or not input values.

virtual const *FormatMetadata* &**getFormatMetadata**() const noexcept = 0

virtual *FormatMetadata* &**getFormatMetadata**() noexcept = 0

virtual bool **equals**(const *RangeTransform* &other) const noexcept = 0

Checks if this equals other.

virtual *BitDepth* **getFileInputBitDepth**() const noexcept = 0

File bit-depth

In a format such as CLF, the range values are scaled to take pixels at the specified inBitDepth to pixels at the specified outBitDepth. This complicates the interpretation of the range values and so this object always holds normalized values and scaling is done on the way from or to file formats such as CLF. Get the bit-depths associated with the range values read from a file or set the bit-depths of values to be written to a file (for file formats such as CLF that support multiple bit-depths).

virtual void **setFileInputBitDepth**(*BitDepth* bitDepth) noexcept = 0

virtual *BitDepth* **getFileOutputBitDepth**() const noexcept = 0

virtual void **setFileOutputBitDepth**(*BitDepth* bitDepth) noexcept = 0

virtual double **getMinInValue**() const noexcept = 0

Get the minimum value for the input.

Range values

These values are normalized relative to what may be stored in file formats such as CLF. For example in a CLF file using a "10i" input depth, a MaxInValue of 1023 in the file is normalized to 1.0. Likewise, for an output depth of "12i", a MaxOutValue of 4095 in the file is normalized to 1.0. The values here are unclamped and may extend outside [0,1].

virtual void **setMinInValue**(double val) noexcept = 0

Set the minimum value for the input.

virtual bool **hasMinInValue**() const noexcept = 0

Is the minimum value for the input set?

virtual void **unsetMinInValue**() noexcept = 0

Unset the minimum value for the input.

virtual void **setMaxInValue**(double val) noexcept = 0

Set the maximum value for the input.

virtual double **getMaxInValue**() const noexcept = 0

Get the maximum value for the input.

virtual bool **hasMaxInValue**() const noexcept = 0

Is the maximum value for the input set?

virtual void **unsetMaxInValue**() noexcept = 0

Unset the maximum value for the input.

virtual void **setMinOutValue**(double val) noexcept = 0

Set the minimum value for the output.

virtual double **getMinOutValue**() const noexcept = 0

Get the minimum value for the output.

virtual bool **hasMinOutValue**() const noexcept = 0

Is the minimum value for the output set?

virtual void **unsetMinOutValue**() noexcept = 0

Unset the minimum value for the output.

virtual void **setMaxOutValue**(double val) noexcept = 0

Set the maximum value for the output.

virtual double **getMaxOutValue**() const noexcept = 0

Get the maximum value for the output.

virtual bool **hasMaxOutValue**() const noexcept = 0

Is the maximum value for the output set?

virtual void **unsetMaxOutValue**() noexcept = 0

Unset the maximum value for the output.

RangeTransform(const *RangeTransform*&) = delete

RangeTransform &**operator=**(const *RangeTransform*&) = delete

virtual **~RangeTransform**() = default

Do not use (needed only for pybind11).

Public Static Functions

static *RangeTransformRcPtr* **Create**()

Creates an instance of RangeTransform.

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *RangeTransform*&) noexcept

typedef std::shared_ptr<const *RangeTransform*> OpenColorIO_v2_5::**ConstRangeTransformRcPtr**

typedef std::shared_ptr<*RangeTransform*> OpenColorIO_v2_5::**RangeTransformRcPtr**

5.7.27 ViewTransform

Python

class PyOpenColorIO.**ViewTransform**

A *:ref:`ViewTransform`* provides a conversion from the main (usually scene-referred) reference space to the display-referred reference space. This allows splitting the conversion from the main reference space to a display into two parts: the ViewTransform plus a display color space.

It is also possible to provide a ViewTransform that converts from the display-referred reference space back to that space. This is useful in cases when a ViewTransform is needed when converting between displays (such as HDR to SDR).

The ReferenceSpaceType indicates whether the ViewTransform converts from scene-to-display reference or display-to-display reference.

The from_reference transform direction is the one that is used when going out towards a display.

ViewTransform(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: PyOpenColorIO.PyOpenColorIO.ViewTransform, referenceSpace: PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType) -> None

2. __init__(self: PyOpenColorIO.PyOpenColorIO.ViewTransform, referenceSpace: PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType = <ReferenceSpaceType.REFERENCE_SPACE_SCENE: 0>, name: str = '', family: str = '', description: str = '', toReference: PyOpenColorIO.PyOpenColorIO.Transform = None, fromReference: PyOpenColorIO.PyOpenColorIO.Transform = None, categories: collections.abc.Sequence[str] = []) -> None

addCategory(*category: str*) → None

clearCategories() → None

getCategories() → *PyOpenColorIO.PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator*

getDescription() → str

getFamily() → str

getInterchangeAttribute(*attrName: str*) → str

Get/Set the interchange attributes.

Currently the only supported attribute name is "amf_transform_ids". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

getInterchangeAttributes() → dict[str, str]

getName() → str

getReferenceSpaceType() → *PyOpenColorIO.PyOpenColorIO.ReferenceSpaceType*

getTransform(*direction:* PyOpenColorIO.PyOpenColorIO.ViewTransformDirection) → *PyOpenColorIO.PyOpenColorIO.Transform*

If a transform in the specified direction has been specified, return it. Otherwise return a null ConstTransformRcPtr

hasCategory(*category: str*) → bool

removeCategory(*category: str*) → None

setDescription(*description: str*) → None

setFamily(*family: str*) → None

setInterchangeAttribute(*attrName: str*, *attrValue: str*) → None

setName(*name: str*) → None

setTransform(*transform:* PyOpenColorIO.PyOpenColorIO.Transform, *direction:* PyOpenColorIO.PyOpenColorIO.ViewTransformDirection) → None

Specify the transform for the appropriate direction. Setting the transform to null will clear it.

class PyOpenColorIO.ViewTransform.**ViewTransformCategoryIterator**

self [*arg0: SupportsInt*] → str

iter(*self*) → *PyOpenColorIO.PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator*

len(*self*) → int

next(*self*) → str

C++

class **ViewTransform**

A *ViewTransform* provides a conversion from the main (usually scene-referred) reference space to the display-referred reference space. This allows splitting the conversion from the main reference space to a display into two parts: the ViewTransform plus a display color space.

It is also possible to provide a ViewTransform that converts from the display-referred reference space back to that space. This is useful in cases when a ViewTransform is needed when converting between displays (such as HDR to SDR).

The ReferenceSpaceType indicates whether the ViewTransform converts from scene-to-display reference or display-to-display reference.

The from_reference transform direction is the one that is used when going out towards a display.

Public Functions

ViewTransformRcPtr **createEditableCopy**() const

const char ***getName**() const noexcept

void **setName**(const char *name) noexcept

const char ***getFamily**() const noexcept

See also:

ColorSpace::getFamily

void **setFamily**(const char *family)

See also:

ColorSpace::setFamily

const char ***getDescription**() const noexcept

void **setDescription**(const char *description)

const char ***getInterchangeAttribute**(const char *attrName) const

Get/Set the interchange attributes.

Currently the only supported attribute name is "amf_transform_ids". Using any other name will throw. If the attribute is not defined, it will return an empty string. Setting the value to an empty string will effectively delete the attribute.

The AMF transform IDs are used to identify specific transforms in the ACES Metadata File. Multiple transform IDs can be specified in a newline-separated string.

void **setInterchangeAttribute**(const char *attrName, const char *value)

std::map<std::string, std::string> **getInterchangeAttributes**() const noexcept

bool **hasCategory**(const char *category) const

See also:

ColorSpace::hasCategory

void **addCategory**(const char *category)

See also:

ColorSpace::addCategory

void **removeCategory**(const char *category)

See also:

ColorSpace::removeCategory

int **getNumCategories**() const

See also:

ColorSpace::getNumCategories

const char ***getCategory**(int index) const

See also:

ColorSpace::getCategory

void **clearCategories**()

See also:

ColorSpace::clearCategories

ReferenceSpaceType **getReferenceSpaceType**() const noexcept

ConstTransformRcPtr **getTransform**(*ViewTransformDirection* dir) const noexcept

If a transform in the specified direction has been specified, return it. Otherwise return a null ConstTransformRcPtr

void **setTransform**(const *ConstTransformRcPtr* &transform, *ViewTransformDirection* dir)

Specify the transform for the appropriate direction. Setting the transform to null will clear it.

ViewTransform(const *ViewTransform*&) = delete

ViewTransform &**operator=**(const *ViewTransform*&) = delete

~ViewTransform()

Do not use (needed only for pybind11).

Public Static Functions

static *ViewTransformRcPtr* **Create**(*ReferenceSpaceType* referenceSpace)

std::ostream &OpenColorIO_v2_5::**operator<<**(std::ostream&, const *ViewTransform*&)

typedef std::shared_ptr<const *ViewTransform*> OpenColorIO_v2_5::**ConstViewTransformRcPtr**

typedef std::shared_ptr<*ViewTransform*> OpenColorIO_v2_5::**ViewTransformRcPtr**

5.8 ASWF

5.8.1 License

OpenColorIO is licensed under the BSD-3-Clause license. Contributions to the library should abide by that license unless otherwised approved by the OCIO TSC and ASWF Governing Board.

See LICENSE on GitHub.

5.8.2 Charter

See Charter.md on GitHub.

Technical Charter (the "Charter") for OpenColorIO a Series of LF Projects, LLC

Adopted _____

This charter (the "Charter") sets forth the responsibilities and procedures for technical contribution to, and oversight of, the OpenColorIO project, which has been established as OpenColorIO a Series of LF Projects, LLC (the "Project"). LF Projects, LLC ("LF Projects") is a Delaware series limited liability company. All Contributors to the Project must comply with the terms of this Charter.

1. Mission and Scope of the Project

- **a.** The mission of the Project is to develop an open source project with the goals indicated in the "README" file within the Project's code repository.
- **b.** The scope of the Project includes software development under an OSI-approved open source license supporting the mission, including documentation, testing, integration and the creation of other artifacts that aid the development, deployment, operation or adoption of the open source software project.

2. Technical Steering Committee

- **a.** The Technical Steering Committee (the "TSC") will be responsible for all technical oversight of the open source Project.
- **b.** The TSC voting members are initially the Project's Committers. At the inception of the project, the Committers of the Project will be as set forth within the "GOVERNANCE" file within the Project's code repository. The TSC may choose an alternative approach for determining the voting members of the TSC, and any such alternative approach will be documented in the GOVERNANCE file. Any meetings of the Technical Steering Committee are intended to be open to the public, and can be conducted electronically, via teleconference, or in person.
- **c.** TSC projects generally will involve Contributors and Committers. The TSC may adopt or modify roles so long as the roles are documented in the GOVERNANCE file. Unless otherwise documented:
  - **i.** Contributors include anyone in the technical community that contributes code, documentation, or other technical artifacts to the Project;
  - **ii.** Committers are Contributors who have earned the ability to modify ("commit") source code, documentation or other technical artifacts in a project's repository; and
  - **iii.** A Contributor may become a Committer by a majority approval of the existing Committers. A Committer may be removed by a majority approval of the other existing Committers.
- **d.** Participation in the Project through becoming a Contributor and Committer is open to anyone so long as they abide by the terms of this Charter.

- **e.** The TSC may (1) establish work flow procedures for the submission, approval, and closure/archiving of projects, (2) set requirements for the promotion of Contributors to Committer status, as applicable, and (3) amend, adjust, refine and/or eliminate the roles of Contributors, and Committers, and create new roles, and publicly document any TSC roles, as it sees fit.
- **f.** The TSC may elect a TSC Chair, who will preside over meetings of the TSC and will serve until their resignation or replacement by the TSC. The TSC Chair, or any other TSC member so designated by the TSC, will serve as the primary communication contact between the Project and the Technical Advisory Council of the Academy Software Foundation of The Linux Foundation.
- **g.** Responsibilities: The TSC will be responsible for all aspects of oversight relating to the Project, which may include:
  - **i.** coordinating the technical direction of the Project;
  - **ii.** approving project or system proposals (including, but not limited to, incubation, deprecation, and changes to a sub-project's scope);
  - **iii.** rganizing sub-projects and removing projects;
  - **ii.** creating sub-committees or working groups to focus on cross-project technical issues and requirements;
  - **v.** appointing representatives to work with other open source or open standards communities;
  - **vi.** establishing community norms, workflows, issuing releases, and security issue reporting policies;
  - **vii.** approving and implementing policies and processes for contributing (to be published in the CONTRIBUTING file) and coordinating with the Series Manager to resolve matters or concerns that may arise as set forth in Section 7 of this Charter;
  - **viii.** discussions, seeking consensus, and where necessary, voting on technical matters relating to the code base that affect multiple projects; and
  - **ix.** coordinating any marketing, events, or communications regarding the Project with the LF Projects Manager or their designee.

3. TSC Voting

- **a.** While the Project aims to operate as a consensus based community, if any TSC decision requires a vote to move the Project forward, the voting members of the TSC will vote on a one vote per voting member basis.
- **b.** Quorum for TSC meetings requires at least two-thirds of all voting members of the TSC to be present. The TSC may continue to meet if quorum is not met, but will be prevented from making any decisions at the meeting.
- **c.** Except as provided in Section 7.c. and 8.a, decisions by vote at a meeting require a majority vote of those in attendance, provided quorum is met. Decisions made by electronic vote without a meeting require a majority vote of all voting members of the TSC.
- **d.** In the event a vote cannot be resolved by the TSC, any voting member of the TSC may refer the matter to the Series Manager for assistance in reaching a resolution.

4. Compliance with Policies

- **a.** This Charter is subject to the Series Agreement for the Project and the Operating Agreement of LF Projects. Contributors will comply with the policies of LF Projects as may be adopted and amended by LF Projects, including, without limitation the policies listed at https://lfprojects.org/policies/.
- **b.** The TSC may adopt a code of conduct ("CoC") for the Project, which is subject to approval by the Series Manager. Contributors to the Project will comply with the CoC or, in the event that a Project-specific CoC has not been approved, the LF Projects Code of Conduct listed at https://lfprojects.org/policies/.
- **c.** When amending or adopting any policy applicable to the Project, LF Projects will publish such policy, as to be amended or adopted, on its web site at least 30 days prior to such policy taking effect; provided, however, that

in the case of any amendment of the Trademark Policy or Terms of Use of LF Projects, any such amendment is effective upon publication on LF Project's web site.

- **d.** All participants must allow open participation from any individual or organization meeting the requirements for contributing under this Charter and any policies adopted for all participants by the TSC, regardless of competitive interests. Put another way, the Project community must not seek to exclude any participant based on any criteria, requirement, or reason other than those that are reasonable and applied on a non-discriminatory basis to all participants in the Project community.
- **e.** The Project will operate in a transparent, open, collaborative, and ethical manner at all times. The output of all Project discussions, proposals, timelines, decisions, and status should be made open and easily visible to all. Any potential violations of this requirement should be reported immediately to the LF Projects Manager.

5. Community Assets

- **a.** LF Projects shall hold title to all trade or service marks used by the Project ("Project Trademarks"), whether based on common law or registered rights. Project Trademarks shall be transferred and assigned to LF Projects to hold on behalf of the Project. Any use of any Project Trademarks by participants in the Project shall be in accordance with the license from LF Projects and inure to the benefit of LF Projects.
- **b.** The Project shall, as permitted and in accordance with such license from LF Projects, develop and own all Project GitHub and social media accounts, and domain name registrations created by the Project community.
- **c.** Under no circumstances shall LF Projects be expected or required to undertake any action on behalf of the Project that is inconsistent with the tax-exempt status or purpose, as applicable, of LFP, Inc. or LF Projects, LLC.

6. General Rules and Operations.

- **a.** The Project will:
  - **i.** engage in the work of the project in a professional manner consistent with maintaining a cohesive community, while also maintaining the goodwill and esteem of LF Projects, LFP, Inc. and other partner organizations in the open source software community; and
  - **ii.** respect the rights of all trademark owners, including any branding and trademark usage guidelines.

7. Intellectual Property Policy

- **a.** Participants acknowledge that the copyright in all new contributions shall be retained by the copyright holder as independent works of authorship and that no contributor or copyright holder will be required to assign copyrights to the Project.
- **b.** Except as described in Section 7.c., all code contributions to the Project are subject to the following:
  - **i.** All new inbound code contributions to the Project must be made using the **BSD 3-Clause License (available here: **https://github.com/AcademySoftwareFoundation/OpenColorIO/blob/main/LICENSE**)** (the "Project License").
  - **ii.** All new inbound code contributions must:
    1. be made pursuant to a duly executed Project Contribution License Agreement (the "CLA"), available on the Project's web site; and
    2. be accompanied by a Developer Certificate of Origin (http://developercertificate.org) sign-off in the source code system that is submitted through a TSC-approved contribution process which will bind the authorized contributor and, if not self-employed, their employer to the applicable license;
  - **iii.** All outbound code will be made available under the Project License.
  - **iv.** Documentation will be received and made available by the Project under the Creative Commons Attribution 4.0 International License (available at http://creativecommons.org/licenses/by/4.0/).
  - **V.** The Project may seek to integrate and contribute back to other open source projects ("Upstream Projects"). In such cases, the Project will conform to all license requirements of the Upstream Projects,

including dependencies, leveraged by the Project. Upstream Project code contributions not stored within the Project's main code repository shall comply with the contribution process and license terms for the applicable Upstream Project.

- **c.** If an alternative inbound or outbound license is required for compliance with the license for a leveraged open source project or is otherwise required to achieve the Project's mission, the Governing Board of the Academy Software Foundation of The Linux Foundation ("Governing Board") or the Governing Board's designated committee may approve the use of an alternative license for specific inbound or outbound contributions on an exception basis. Any exceptions must be approved by a vote of the Governing Board and must be limited in scope to what is required for such purpose. To request an exception, please describe the contribution, the alternative open source license(s), and the justification for using an alternative open source license for the Project.
- **d.** Contributed files should contain license information, such as SPDX short form identifiers, indicating the open source license or licenses pertaining to the file.

8. Amendments

- **a.** This charter may be amended by a two-thirds vote of the entire TSC and is subject to approval by LF Projects.

5.8.3 CLA and DCO

OpenColorIO is a project hosted by the Academy Software Foundation (ASWF) and follows the open source software best practice policies of the ASWF TAC with the guidance from the Linux Foundation.

See CONTRIBUTING.md for more information.

CLA

Developers who wish to contribute code to be considered for inclusion in OpenColorIO (OCIO) must first complete a Contributor License Agreement (CLA).

OCIO uses EasyCLA for managing CLAs, which automatically checks to ensure CLAs are signed by a contributor before a commit can be merged.

If you are an individual writing the code on your own time and you're SURE you are the sole owner of any intellectual property you contribute, you can *sign the CLA as an individual contributor*.

If you are writing the code as part of your job, or if there is any possibility that your employers might think they own any intellectual property you create, then you should use the *Corporate Contributor Licence Agreement*.

The OCIO CLA's are the standard forms used by Linux Foundation projects and recommended by the ASWF TAC.

CLA-corporate

See CLA-corporate.md on GitHub.

Corporate Contributor License Agreement ("Agreement")

Thank you for your interest in the OpenColorIO Project a Series of LF Projects, LLC (hereinafter "Project"). In order to clarify the intellectual property licenses granted with Contributions from any corporate entity to the Project, LF Projects, LLC ("LF Projects") is required to have a Corporate Contributor License Agreement (CCLA) on file that has been signed by each contributing corporation.

Each contributing corporation ("You") must accept and agree that, for any Contribution (as defined below), You and all other individuals and entities that control You, are controlled by You, or are under common control with You, are bound by the licenses granted and representations made as though all such individuals and entities are a single contributor. For

the purposes of this definition, “control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

“Contribution” means any code, documentation or other original work of authorship that is submitted to LF Projects for inclusion in the Project by Your employee or by any individual or entity that controls You, or is under common control with You or is controlled by You and authorized to make the submission on Your behalf.

You accept and agree that all of Your present and future Contributions to the Project shall be:

Submitted under a Developer’s Certificate of Origin v. 1.1 (DCO); and Licensed under the BSD-3-Clause License.

You agree that You shall be bound by the terms of the BSD-3-Clause License for all contributions made by You and Your employees. Your designated employees are those listed by Your CLA Manager(s) on the system of record for the Project. You agree to identify Your initial CLA Manager below and thereafter maintain current CLA Manager records in the Project’s system of record.

Initial CLA Manager (Name and Email): _____

Corporate Signature:

Company Name: _____

Signature: _____

Name: _____

Title _____

Date: _____

CLA-individual

See CLA-individual.md on GitHub.

Individual Contributor License Agreement (“Agreement”)

Thank you for your interest in the OpenColorIO Project a Series of LF Projects, LLC (hereinafter “Project”). In order to clarify the intellectual property licenses granted with Contributions from any corporate entity to the Project, LF Projects, LLC (“LF Projects”) is required to have an Individual Contributor License Agreement (ICLA) on file that has been signed by each contributing individual. (For legal entities, please use the Corporate Contributor License Agreement (CCLA).)

Each contributing individual (“You”) must accept and agree that, for any Contribution (as defined below), You are bound by the licenses granted and representations made herein.

“Contribution” means any code, documentation or other original work of authorship that is submitted to LF Projects for inclusion in the Project by You or by another person authorized to make the submission on Your behalf.

You accept and agree that all of Your present and future Contributions to the Project shall be:

Submitted under a Developer’s Certificate of Origin v. 1.1 (DCO); and Licensed under the BSD-3-Clause License.

Signature: _____

Name: _____

Date: _____

Commit Sign-Off

Every commit must be signed off. That is, every commit log message must include a "Signed-off-by" line (generated, for example, with "git commit –signoff" or "git commit -s"), indicating that the committer wrote the code and has the right to release it under the Modified-BSD-3-Clause license.

Here is an example Signed-off-by line, which indicates that the submitter accepts the DCO:

```
Signed-off-by: John Doe <john.doe@example.com>
```

If John Doe has signed an individual CLA, or his corporation's CLA Manager has included his GitHub account in a corporate CLA approved list, his pull request can be merged. Otherwise the EasyCLA system will provide instructions on signing a CLA, or request inclusion in an existing corporate CLA approved list.

See the ASWF TAC CONTRIBUTING.md file for more information on this requirement.

DCO

https://developercertificate.org/

5.8.4 ASWF Docker

OpenColorIO's CI infrastructure utilizes ASWF-managed docker containers for Linux builds. Each image implements a specific VFX Reference Platform calendar year and all dependencies for a specific ASWF project.

For example, the aswf/ci-ocio:2020 container provides a build environment with all upstream OpenColorIO dependencies and adheres to VFX Reference Platform CY2020. VFX Reference Platform calendar years starting with 2018 up to the current draft year specification are supported.

These Docker images are available in the aswf DockerHub repository for public use. See the table in the aswf-docker source GitHub repository for a summary of all available images.

INDEX

Symbols

__and__() (*PyOpenColorIO.ColorSpaceSet method*), 206
__contains__() (*PyOpenColorIO.FormatMetadata method*), 300
__eq__() (*PyOpenColorIO.ColorSpaceSet method*), 206
__getitem__() (*PyOpenColorIO.Baker.FormatIterator method*), 196
__getitem__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator method*), 392
__getitem__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinStyleIterator method*), 392
__getitem__() (*PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator method*), 201
__getitem__() (*PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator method*), 201
__getitem__() (*PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator method*), 181
__getitem__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator method*), 208
__getitem__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator method*), 208
__getitem__() (*PyOpenColorIO.Config.ActiveColorSpaceIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.ActiveColorSpaceNameIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.ActiveNamedTransformIterator method*), 232
__getitem__() (*PyOpenColorIO.Config.ActiveNamedTransformNameIterator method*), 232
__getitem__() (*PyOpenColorIO.Config.ColorSpaceIterator method*), 229
__getitem__() (*PyOpenColorIO.Config.ColorSpaceNameIterator method*), 229
__getitem__() (*PyOpenColorIO.Config.DisplayIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.EnvironmentVarNameIterator method*), 229
__getitem__() (*PyOpenColorIO.Config.LookIterator method*), 231
__getitem__() (*PyOpenColorIO.Config.LookNameIterator method*), 231
__getitem__() (*PyOpenColorIO.Config.NamedTransformIterator method*), 232
__getitem__() (*PyOpenColorIO.Config.NamedTransformNameIterator method*), 231
__getitem__() (*PyOpenColorIO.Config.RoleColorSpaceIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.RoleNameIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.SearchPathIterator method*), 229
__getitem__() (*PyOpenColorIO.Config.SharedViewIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.ViewForColorSpaceIterator method*), 231
__getitem__() (*PyOpenColorIO.Config.ViewIterator method*), 230
__getitem__() (*PyOpenColorIO.Config.ViewTransformIterator method*), 231
__getitem__() (*PyOpenCol-*

orIO.Config.ViewTransformNameIterator method), 231
__getitem__() (PyOpenColorIO.Context.SearchPathIterator method), 259
__getitem__() (PyOpenColorIO.Context.StringVarIterator method), 258
__getitem__() (PyOpenColorIO.Context.StringVarNameIterator method), 258
__getitem__() (PyOpenColorIO.FileTransform.FormatIterator method), 409
__getitem__() (PyOpenColorIO.FormatMetadata method), 300
__getitem__() (PyOpenColorIO.FormatMetadata.AttributeIterator method), 301
__getitem__() (PyOpenColorIO.FormatMetadata.AttributeNameIterator method), 301
__getitem__() (PyOpenColorIO.FormatMetadata.ChildElementIterator method), 301
__getitem__() (PyOpenColorIO.FormatMetadata.ConstChildElementIterator method), 301
__getitem__() (PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator method), 369
__getitem__() (PyOpenColorIO.GpuShaderDesc.DynamicPropertyIterator method), 382
__getitem__() (PyOpenColorIO.GpuShaderDesc.Texture3DIterator method), 382
__getitem__() (PyOpenColorIO.GpuShaderDesc.TextureIterator method), 381
__getitem__() (PyOpenColorIO.GpuShaderDesc.UniformIterator method), 381
__getitem__() (PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator method), 328
__getitem__() (PyOpenColorIO.GroupTransform.TransformIterator method), 414
__getitem__() (PyOpenColorIO.GroupTransform.WriteFormatIterator method), 414
__getitem__() (PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator method), 192
__getitem__() (PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator method), 192
__getitem__() (PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator method), 344
__getitem__() (PyOpenColorIO.Processor.TransformFormatMetadataIterator method), 349
__getitem__() (PyOpenColorIO.ProcessorMetadata.FileIterator method), 356
__getitem__() (PyOpenColorIO.ProcessorMetadata.LookIterator method), 356
__getitem__() (PyOpenColorIO.SystemMonitors.MonitorIterator method), 387
__getitem__() (PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator method), 438
__getitem__() (PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator method), 364
__getitem__() (PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator method), 364
__init__() (PyOpenColorIO.AllocationTransform method), 389
__init__() (PyOpenColorIO.Baker method), 194
__init__() (PyOpenColorIO.BuiltinTransform method), 391
__init__() (PyOpenColorIO.BuiltinTransformRegistry method), 392
__init__() (PyOpenColorIO.CDLTransform method), 394
__init__() (PyOpenColorIO.CPUProcessor method), 351
__init__() (PyOpenColorIO.ColorSpace method), 198
__init__() (PyOpenColorIO.ColorSpaceMenuHelper method), 181
__init__() (PyOpenColorIO.ColorSpaceMenuParameters method), 179
__init__() (PyOpenColorIO.ColorSpaceSet method), 206
__init__() (PyOpenColorIO.ColorSpaceTransform method), 397
__init__() (PyOpenColorIO.Config method), 215
__init__() (PyOpenColorIO.Config.ActiveDisplaysListIterator method), 210
__init__() (PyOpenCol-

orIO.Config.ActiveViewsListIterator method), 210
__init__() (PyOpenColorIO.Config.DisplayAllIterator method), 212
__init__() (PyOpenColorIO.Config.ViewForViewTypeIterator method), 215
__init__() (PyOpenColorIO.Config.VirtualViewIterator method), 215
__init__() (PyOpenColorIO.Context method), 257
__init__() (PyOpenColorIO.DisplayViewTransform method), 398
__init__() (PyOpenColorIO.DynamicProperty method), 262
__init__() (PyOpenColorIO.ExponentTransform method), 400
__init__() (PyOpenColorIO.ExponentWithLinearTransform method), 402
__init__() (PyOpenColorIO.ExposureContrastTransform method), 405
__init__() (PyOpenColorIO.FileRules method), 358
__init__() (PyOpenColorIO.FileTransform method), 408
__init__() (PyOpenColorIO.FixedFunctionTransform method), 411
__init__() (PyOpenColorIO.FormatMetadata method), 300
__init__() (PyOpenColorIO.GPUProcessor method), 355
__init__() (PyOpenColorIO.GpuShaderDesc method), 378
__init__() (PyOpenColorIO.GpuShaderDesc.TextureDimensions method), 377
__init__() (PyOpenColorIO.GradingBSplineCurve method), 327
__init__() (PyOpenColorIO.GradingControlPoint method), 326
__init__() (PyOpenColorIO.GradingHueCurve method), 324
__init__() (PyOpenColorIO.GradingHueCurveTransform method), 321
__init__() (PyOpenColorIO.GradingPrimary method), 313
__init__() (PyOpenColorIO.GradingPrimaryTransform method), 311
__init__() (PyOpenColorIO.GradingRGBCurve method), 319
__init__() (PyOpenColorIO.GradingRGBCurveTransform method), 316
__init__() (PyOpenColorIO.GradingRGBM method), 315
__init__() (PyOpenColorIO.GradingRGBMSW method), 333
__init__() (PyOpenColorIO.GradingTone method), 332
__init__() (PyOpenColorIO.GradingToneTransform method), 329
__init__() (PyOpenColorIO.GroupTransform method), 412
__init__() (PyOpenColorIO.ImageDesc method), 334
__init__() (PyOpenColorIO.LegacyViewingPipeline method), 188
__init__() (PyOpenColorIO.LogAffineTransform method), 415
__init__() (PyOpenColorIO.LogCameraTransform method), 418
__init__() (PyOpenColorIO.LogTransform method), 420
__init__() (PyOpenColorIO.Look method), 341
__init__() (PyOpenColorIO.LookTransform method), 422
__init__() (PyOpenColorIO.Lut1DTransform method), 424
__init__() (PyOpenColorIO.Lut3DTransform method), 427
__init__() (PyOpenColorIO.MatrixTransform method), 430
__init__() (PyOpenColorIO.MixingColorSpaceManager method), 191
__init__() (PyOpenColorIO.MixingSlider method), 191
__init__() (PyOpenColorIO.NamedTransform method), 343
__init__() (PyOpenColorIO.NamedTransform.NamedTransformAliasIterator method), 343
__init__() (PyOpenColorIO.PackedImageDesc method), 336
__init__() (PyOpenColorIO.PlanarImageDesc method), 339
__init__() (PyOpenColorIO.Processor method), 347
__init__() (PyOpenColorIO.ProcessorMetadata method), 356
__init__() (PyOpenColorIO.RangeTransform method), 433
__init__() (PyOpenColorIO.SystemMonitors method), 387
__init__() (PyOpenColorIO.Transform method), 388
__init__() (PyOpenColorIO.ViewTransform method), 437

__init__() (*PyOpenColorIO.ViewingRules method*), 363
__iter__() (*PyOpenColorIO.Baker.FormatIterator method*), 196
__iter__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator method*), 392
__iter__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinStyleIterator method*), 392
__iter__() (*PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator method*), 202
__iter__() (*PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator method*), 201
__iter__() (*PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator method*), 181
__iter__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator method*), 208
__iter__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator method*), 208
__iter__() (*PyOpenColorIO.Config.ActiveColorSpaceIterator method*), 230
__iter__() (*PyOpenColorIO.Config.ActiveColorSpaceNameIterator method*), 230
__iter__() (*PyOpenColorIO.Config.ActiveNamedTransformIterator method*), 232
__iter__() (*PyOpenColorIO.Config.ActiveNamedTransformNameIterator method*), 232
__iter__() (*PyOpenColorIO.Config.ColorSpaceIterator method*), 229
__iter__() (*PyOpenColorIO.Config.ColorSpaceNameIterator method*), 229
__iter__() (*PyOpenColorIO.Config.DisplayIterator method*), 230
__iter__() (*PyOpenColorIO.Config.EnvironmentVarNameIterator method*), 229
__iter__() (*PyOpenColorIO.Config.LookIterator method*), 231
__iter__() (*PyOpenColorIO.Config.LookNameIterator method*), 231
__iter__() (*PyOpenColorIO.Config.NamedTransformIterator method*), 232
__iter__() (*PyOpenColorIO.Config.NamedTransformNameIterator method*), 231
__iter__() (*PyOpenColorIO.Config.RoleColorSpaceIterator method*), 230
__iter__() (*PyOpenColorIO.Config.RoleNameIterator method*), 230
__iter__() (*PyOpenColorIO.Config.SearchPathIterator method*), 229
__iter__() (*PyOpenColorIO.Config.SharedViewIterator method*), 230
__iter__() (*PyOpenColorIO.Config.ViewForColorSpaceIterator method*), 231
__iter__() (*PyOpenColorIO.Config.ViewIterator method*), 231
__iter__() (*PyOpenColorIO.Config.ViewTransformIterator method*), 231
__iter__() (*PyOpenColorIO.Config.ViewTransformNameIterator method*), 231
__iter__() (*PyOpenColorIO.Context.SearchPathIterator method*), 259
__iter__() (*PyOpenColorIO.Context.StringVarIterator method*), 258
__iter__() (*PyOpenColorIO.Context.StringVarNameIterator method*), 258
__iter__() (*PyOpenColorIO.FileTransform.FormatIterator method*), 409
__iter__() (*PyOpenColorIO.FormatMetadata method*), 300
__iter__() (*PyOpenColorIO.FormatMetadata.AttributeIterator method*), 301
__iter__() (*PyOpenColorIO.FormatMetadata.AttributeNameIterator method*), 301
__iter__() (*PyOpenColorIO.FormatMetadata.ChildElementIterator method*), 301
__iter__() (*PyOpenColorIO.FormatMetadata.ConstChildElementIterator method*), 301
__iter__() (*PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator method*), 369

__iter__() (PyOpenColorIO.GpuShaderDesc.DynamicPropertyIterator method), 382
__iter__() (PyOpenColorIO.GpuShaderDesc.Texture3DIterator method), 382
__iter__() (PyOpenColorIO.GpuShaderDesc.TextureIterator method), 381
__iter__() (PyOpenColorIO.GpuShaderDesc.UniformIterator method), 381
__iter__() (PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator method), 328
__iter__() (PyOpenColorIO.GroupTransform.TransformIterator method), 414
__iter__() (PyOpenColorIO.GroupTransform.WriteFormatIterator method), 414
__iter__() (PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator method), 192
__iter__() (PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator method), 192
__iter__() (PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator method), 344
__iter__() (PyOpenColorIO.Processor.TransformFormatMetadataIterator method), 349
__iter__() (PyOpenColorIO.ProcessorMetadata.FileIterator method), 356
__iter__() (PyOpenColorIO.ProcessorMetadata.LookIterator method), 356
__iter__() (PyOpenColorIO.SystemMonitors.MonitorIterator method), 387
__iter__() (PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator method), 438
__iter__() (PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator method), 364
__iter__() (PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator method), 364
__len__() (PyOpenColorIO.Baker.FormatIterator method), 196
__len__() (PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator method), 392
__len__() (PyOpenColorIO.BuiltinTransformRegistry.BuiltinStyleIterator method), 392
__len__() (PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator method), 202
__len__() (PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator method), 201
__len__() (PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator method), 181
__len__() (PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator method), 208
__len__() (PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator method), 208
__len__() (PyOpenColorIO.Config.ActiveColorSpaceIterator method), 230
__len__() (PyOpenColorIO.Config.ActiveColorSpaceNameIterator method), 230
__len__() (PyOpenColorIO.Config.ActiveNamedTransformIterator method), 232
__len__() (PyOpenColorIO.Config.ActiveNamedTransformNameIterator method), 232
__len__() (PyOpenColorIO.Config.ColorSpaceIterator method), 229
__len__() (PyOpenColorIO.Config.ColorSpaceNameIterator method), 229
__len__() (PyOpenColorIO.Config.DisplayIterator method), 230
__len__() (PyOpenColorIO.Config.EnvironmentVarNameIterator method), 229
__len__() (PyOpenColorIO.Config.LookIterator method), 231
__len__() (PyOpenColorIO.Config.LookNameIterator method), 231
__len__() (PyOpenColorIO.Config.NamedTransformIterator method), 232
__len__() (PyOpenColorIO.Config.NamedTransformNameIterator method), 232
__len__() (PyOpenColorIO.Config.RoleColorSpaceIterator method),

230
__len__() (*PyOpenColorIO.Config.RoleNameIterator method*), 230
__len__() (*PyOpenColorIO.Config.SearchPathIterator method*), 229
__len__() (*PyOpenColorIO.Config.SharedViewIterator method*), 230
__len__() (*PyOpenColorIO.Config.ViewForColorSpaceIterator method*), 231
__len__() (*PyOpenColorIO.Config.ViewIterator method*), 231
__len__() (*PyOpenColorIO.Config.ViewTransformIterator method*), 231
__len__() (*PyOpenColorIO.Config.ViewTransformNameIterator method*), 231
__len__() (*PyOpenColorIO.Context.SearchPathIterator method*), 259
__len__() (*PyOpenColorIO.Context.StringVarIterator method*), 258
__len__() (*PyOpenColorIO.Context.StringVarNameIterator method*), 258
__len__() (*PyOpenColorIO.FileTransform.FormatIterator method*), 409
__len__() (*PyOpenColorIO.FormatMetadata method*), 300
__len__() (*PyOpenColorIO.FormatMetadata.AttributeIterator method*), 301
__len__() (*PyOpenColorIO.FormatMetadata.AttributeNameIterator method*), 301
__len__() (*PyOpenColorIO.FormatMetadata.ChildElementIterator method*), 301
__len__() (*PyOpenColorIO.FormatMetadata.ConstChildElementIterator method*), 301
__len__() (*PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator method*), 370
__len__() (*PyOpenColorIO.GpuShaderDesc.DynamicPropertyIterator method*), 382
__len__() (*PyOpenColorIO.GpuShaderDesc.Texture3DIterator method*), 382
__len__() (*PyOpenColorIO.GpuShaderDesc.TextureIterator method*), 381
__len__() (*PyOpenColorIO.GpuShaderDesc.UniformIterator method*), 381
__len__() (*PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator method*), 328
__len__() (*PyOpenColorIO.GroupTransform.TransformIterator method*), 414
__len__() (*PyOpenColorIO.GroupTransform.WriteFormatIterator method*), 414
__len__() (*PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator method*), 192
__len__() (*PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator method*), 192
__len__() (*PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator method*), 344
__len__() (*PyOpenColorIO.Processor.TransformFormatMetadataIterator method*), 349
__len__() (*PyOpenColorIO.ProcessorMetadata.FileIterator method*), 356
__len__() (*PyOpenColorIO.ProcessorMetadata.LookIterator method*), 356
__len__() (*PyOpenColorIO.SystemMonitors.MonitorIterator method*), 387
__len__() (*PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator method*), 438
__len__() (*PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator method*), 364
__len__() (*PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator method*), 364
__ne__() (*PyOpenColorIO.ColorSpaceSet method*), 206
__next__() (*PyOpenColorIO.Baker.FormatIterator method*), 196
__next__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinIterator method*), 393
__next__() (*PyOpenColorIO.BuiltinTransformRegistry.BuiltinStyleIterator method*), 392
__next__() (*PyOpenColorIO.ColorSpace.ColorSpaceAliasIterator*

method), 202
__next__() (*PyOpenColorIO.ColorSpace.ColorSpaceCategoryIterator method*), 201
__next__() (*PyOpenColorIO.ColorSpaceMenuHelper.ColorSpaceLevelIterator method*), 181
__next__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceIterator method*), 208
__next__() (*PyOpenColorIO.ColorSpaceSet.ColorSpaceNameIterator method*), 208
__next__() (*PyOpenColorIO.Config.ActiveColorSpaceIterator method*), 230
__next__() (*PyOpenColorIO.Config.ActiveColorSpaceNameIterator method*), 230
__next__() (*PyOpenColorIO.Config.ActiveNamedTransformIterator method*), 232
__next__() (*PyOpenColorIO.Config.ActiveNamedTransformNameIterator method*), 232
__next__() (*PyOpenColorIO.Config.ColorSpaceIterator method*), 229
__next__() (*PyOpenColorIO.Config.ColorSpaceNameIterator method*), 229
__next__() (*PyOpenColorIO.Config.DisplayIterator method*), 230
__next__() (*PyOpenColorIO.Config.EnvironmentVarNameIterator method*), 229
__next__() (*PyOpenColorIO.Config.LookIterator method*), 231
__next__() (*PyOpenColorIO.Config.LookNameIterator method*), 231
__next__() (*PyOpenColorIO.Config.NamedTransformIterator method*), 232
__next__() (*PyOpenColorIO.Config.NamedTransformNameIterator method*), 232
__next__() (*PyOpenColorIO.Config.RoleColorSpaceIterator method*), 230
__next__() (*PyOpenColorIO.Config.RoleNameIterator method*), 230
__next__() (*PyOpenColorIO.Config.SearchPathIterator method*), 229
__next__() (*PyOpenColorIO.Config.SharedViewIterator method*), 230
__next__() (*PyOpenColorIO.Config.ViewForColorSpaceIterator method*), 231
__next__() (*PyOpenColorIO.Config.ViewIterator method*), 231
__next__() (*PyOpenColorIO.Config.ViewTransformIterator method*), 231
__next__() (*PyOpenColorIO.Config.ViewTransformNameIterator method*), 231
__next__() (*PyOpenColorIO.Context.SearchPathIterator method*), 259
__next__() (*PyOpenColorIO.Context.StringVarIterator method*), 259
__next__() (*PyOpenColorIO.Context.StringVarNameIterator method*), 258
__next__() (*PyOpenColorIO.FileTransform.FormatIterator method*), 409
__next__() (*PyOpenColorIO.FormatMetadata.AttributeIterator method*), 301
__next__() (*PyOpenColorIO.FormatMetadata.AttributeNameIterator method*), 301
__next__() (*PyOpenColorIO.FormatMetadata.ChildElementIterator method*), 301
__next__() (*PyOpenColorIO.FormatMetadata.ConstChildElementIterator method*), 301
__next__() (*PyOpenColorIO.GpuShaderCreator.DynamicPropertyIterator method*), 370
__next__() (*PyOpenColorIO.GpuShaderDesc.DynamicPropertyIterator method*), 382
__next__() (*PyOpenColorIO.GpuShaderDesc.Texture3DIterator method*), 382
__next__() (*PyOpenColorIO.GpuShaderDesc.TextureIterator method*), 382
__next__() (*PyOpenColorIO.GpuShaderDesc.UniformIterator method*), 381
__next__() (*PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator*

method), 328
__next__() (*PyOpenColorIO.GroupTransform.TransformIterator method*), 414
__next__() (*PyOpenColorIO.GroupTransform.WriteFormatIterator method*), 414
__next__() (*PyOpenColorIO.MixingColorSpaceManager.MixingEncodingIterator method*), 192
__next__() (*PyOpenColorIO.MixingColorSpaceManager.MixingSpaceIterator method*), 192
__next__() (*PyOpenColorIO.NamedTransform.NamedTransformCategoryIterator method*), 344
__next__() (*PyOpenColorIO.Processor.TransformFormatMetadataIterator method*), 349
__next__() (*PyOpenColorIO.ProcessorMetadata.FileIterator method*), 356
__next__() (*PyOpenColorIO.ProcessorMetadata.LookIterator method*), 356
__next__() (*PyOpenColorIO.SystemMonitors.MonitorIterator method*), 387
__next__() (*PyOpenColorIO.ViewTransform.ViewTransformCategoryIterator method*), 438
__next__() (*PyOpenColorIO.ViewingRules.ViewingRuleColorSpaceIterator method*), 364
__next__() (*PyOpenColorIO.ViewingRules.ViewingRuleEncodingIterator method*), 364
__or__() (*PyOpenColorIO.ColorSpaceSet method*), 206
__setitem__() (*PyOpenColorIO.FormatMetadata method*), 300
__setitem__() (*PyOpenColorIO.GradingBSplineCurve.GradingControlPointIterator method*), 328
__str__() (*PyOpenColorIO.Config method*), 215
__sub__() (*PyOpenColorIO.ColorSpaceSet method*), 206

A

ActiveColorSpaceIterator (*class in PyOpenColorIO.Config*), 230
ActiveColorSpaceNameIterator (*class in PyOpenColorIO.Config*), 229
ActiveNamedTransformIterator (*class in PyOpenColorIO.Config*), 232
ActiveNamedTransformNameIterator (*class in PyOpenColorIO.Config*), 232
add3DTexture() (*PyOpenColorIO.GpuShaderDesc method*), 378
addActiveDisplay() (*PyOpenColorIO.Config method*), 215
addActiveView() (*PyOpenColorIO.Config method*), 215
addAlias() (*PyOpenColorIO.ColorSpace method*), 198
addAlias() (*PyOpenColorIO.NamedTransform method*), 343
addCategory() (*PyOpenColorIO.ColorSpace method*), 199
addCategory() (*PyOpenColorIO.NamedTransform method*), 343
addCategory() (*PyOpenColorIO.ViewTransform method*), 438
addChildElement() (*PyOpenColorIO.FormatMetadata method*), 300
AddColorSpace() (*in module PyOpenColorIO.ColorSpaceHelpers*), 177
addColorSpace() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
addColorSpace() (*PyOpenColorIO.ColorSpaceSet method*), 207
addColorSpace() (*PyOpenColorIO.Config method*), 215
addColorSpace() (*PyOpenColorIO.ViewingRules method*), 363
addColorSpaces() (*PyOpenColorIO.ColorSpaceSet method*), 207
addDisplaySharedView() (*PyOpenColorIO.Config method*), 216
AddDisplayView() (*in module PyOpenColorIO.DisplayViewHelpers*), 186
addDisplayView() (*PyOpenColorIO.Config method*), 216
addEncoding() (*PyOpenColorIO.ViewingRules method*), 363
addEnvironmentVar() (*PyOpenColorIO.Config method*), 216
addFile() (*PyOpenColorIO.ProcessorMetadata method*), 356
addLook() (*PyOpenColorIO.Config method*), 216
addLook() (*PyOpenColorIO.ProcessorMetadata method*), 356
addNamedTransform() (*PyOpenColorIO.Config method*), 216
addSearchPath() (*PyOpenColorIO.Config method*), 216
addSearchPath() (*PyOpenColorIO.Context method*), 257
addSharedView() (*PyOpenColorIO.Config method*),

216
addTexture() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToFunctionFooterShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 367
addToFunctionFooterShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToFunctionHeaderShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 367
addToFunctionHeaderShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToFunctionShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 367
addToFunctionShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToHelperShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 368
addToHelperShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToParameterDeclareShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 368
addToParameterDeclareShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addToTextureDeclareShaderCode() (*PyOpenColorIO.GpuShaderCreator method*), 368
addToTextureDeclareShaderCode() (*PyOpenColorIO.GpuShaderDesc method*), 378
addViewTransform() (*PyOpenColorIO.Config method*), 216
addVirtualDisplaySharedView() (*PyOpenColorIO.Config method*), 216
addVirtualDisplayView() (*PyOpenColorIO.Config method*), 216
Allocation (*class in PyOpenColorIO*), 280
ALLOCATION_LG2 (*PyOpenColorIO.Allocation attribute*), 280
ALLOCATION_UNIFORM (*PyOpenColorIO.Allocation attribute*), 280
ALLOCATION_UNKNOWN (*PyOpenColorIO.Allocation attribute*), 280
AllocationFromString() (*in module PyOpenColorIO*), 308
AllocationToString() (*in module PyOpenColorIO*), 308
AllocationTransform (*class in PyOpenColorIO*), 389
appendTransform() (*PyOpenColorIO.GroupTransform method*), 413
apply() (*PyOpenColorIO.CPUProcessor method*), 351
applyRGB() (*PyOpenColorIO.CPUProcessor method*), 352
applyRGBA() (*PyOpenColorIO.CPUProcessor method*), 352
archive() (*PyOpenColorIO.Config method*), 216
AreViewsEqual() (*PyOpenColorIO.Config static method*), 210
AreVirtualViewsEqual() (*PyOpenColorIO.Config static method*), 211
AttributeIterator (*class in PyOpenColorIO.FormatMetadata*), 301
AttributeNameIterator (*class in PyOpenColorIO.FormatMetadata*), 301
AutoStride (*in module PyOpenColorIO*), 335

B

bake() (*PyOpenColorIO.Baker method*), 194
Baker (*class in PyOpenColorIO*), 194
begin() (*PyOpenColorIO.GpuShaderCreator method*), 368
begin() (*PyOpenColorIO.GpuShaderDesc method*), 378
BIT_DEPTH_F16 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_F32 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT10 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT12 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT14 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT16 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT32 (*PyOpenColorIO.BitDepth attribute*), 277
BIT_DEPTH_UINT8 (*PyOpenColorIO.BitDepth attribute*), 278
BIT_DEPTH_UNKNOWN (*PyOpenColorIO.BitDepth attribute*), 278
BitDepth (*class in PyOpenColorIO*), 277
BitDepthFromString() (*in module PyOpenColorIO*), 308
BitDepthIsFloat() (*in module PyOpenColorIO*), 308
BitDepthToInt() (*in module PyOpenColorIO*), 308
BitDepthToString() (*in module PyOpenColorIO*), 308
blacks (*PyOpenColorIO.GradingTone property*), 332
blue (*PyOpenColorIO.GradingRGBCurve property*), 319
blue (*PyOpenColorIO.GradingRGBM property*), 315
blue (*PyOpenColorIO.GradingRGBMSW property*), 333
BoolFromString() (*in module PyOpenColorIO*), 307
BoolToString() (*in module PyOpenColorIO*), 307
brightness (*PyOpenColorIO.GradingPrimary property*), 313
bufferOffset (*PyOpenColorIO.GpuShaderDesc.UniformData property*), 380
BuiltinIterator (*class in PyOpenColorIO.BuiltinTransformRegistry*), 392

BuiltinStyleIterator (*class in PyOpenColorIO.BuiltinTransformRegistry*), 392
BuiltinTransform (*class in PyOpenColorIO*), 391
BuiltinTransformRegistry (*class in PyOpenColorIO*), 392

C

CDL_ASC (*PyOpenColorIO.CDLStyle attribute*), 288
CDL_NO_CLAMP (*PyOpenColorIO.CDLStyle attribute*), 288
CDL_TRANSFORM_DEFAULT (*PyOpenColorIO.CDLStyle attribute*), 288
CDLStyle (*class in PyOpenColorIO*), 288
CDLStyleFromString() (*in module PyOpenColorIO*), 309
CDLStyleToString() (*in module PyOpenColorIO*), 309
CDLTransform (*class in PyOpenColorIO*), 393
channel (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
CHANNEL_ORDERING_ABGR (*PyOpenColorIO.ChannelOrdering attribute*), 279
CHANNEL_ORDERING_BGR (*PyOpenColorIO.ChannelOrdering attribute*), 279
CHANNEL_ORDERING_BGRA (*PyOpenColorIO.ChannelOrdering attribute*), 279
CHANNEL_ORDERING_RGB (*PyOpenColorIO.ChannelOrdering attribute*), 279
CHANNEL_ORDERING_RGBA (*PyOpenColorIO.ChannelOrdering attribute*), 279
ChannelOrdering (*class in PyOpenColorIO*), 279
ChildElementIterator (*class in PyOpenColorIO.FormatMetadata*), 301
clampBlack (*PyOpenColorIO.GradingPrimary property*), 313
clampWhite (*PyOpenColorIO.GradingPrimary property*), 313
clear() (*PyOpenColorIO.FormatMetadata method*), 300
clearActiveDisplays() (*PyOpenColorIO.Config method*), 217
clearActiveViews() (*PyOpenColorIO.Config method*), 217
clearAddedColorSpaces() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
clearAliases() (*PyOpenColorIO.ColorSpace method*), 199
clearAliases() (*PyOpenColorIO.NamedTransform method*), 344
ClearAllCaches() (*in module PyOpenColorIO*), 303
clearCategories() (*PyOpenColorIO.ColorSpace method*), 199
clearCategories() (*PyOpenColorIO.NamedTransform method*), 344
clearCategories() (*PyOpenColorIO.ViewTransform method*), 438
clearColorSpaces() (*PyOpenColorIO.ColorSpaceSet method*), 207
clearColorSpaces() (*PyOpenColorIO.Config method*), 217
clearDisplays() (*PyOpenColorIO.Config method*), 217
clearEnvironmentVars() (*PyOpenColorIO.Config method*), 217
clearLooks() (*PyOpenColorIO.Config method*), 217
clearNamedTransforms() (*PyOpenColorIO.Config method*), 217
clearProcessorCache() (*PyOpenColorIO.Config method*), 217
clearSearchPaths() (*PyOpenColorIO.Config method*), 217
clearSearchPaths() (*PyOpenColorIO.Context method*), 257
clearSharedViews() (*PyOpenColorIO.Config method*), 217
clearStringVars() (*PyOpenColorIO.Context method*), 257
clearViewTransforms() (*PyOpenColorIO.Config method*), 217
clearVirtualDisplay() (*PyOpenColorIO.Config method*), 217
clone() (*PyOpenColorIO.GpuShaderCreator method*), 368
clone() (*PyOpenColorIO.GpuShaderDesc method*), 378
ColorSpace (*class in PyOpenColorIO*), 198
COLORSPACE_ACTIVE (*PyOpenColorIO.ColorSpaceVisibility attribute*), 270
COLORSPACE_ALL (*PyOpenColorIO.ColorSpaceVisibility attribute*), 270
COLORSPACE_DIR_FROM_REFERENCE (*PyOpenColorIO.ColorSpaceDirection attribute*), 271
COLORSPACE_DIR_TO_REFERENCE (*PyOpenColorIO.ColorSpaceDirection attribute*), 271
COLORSPACE_INACTIVE (*PyOpenColorIO.ColorSpaceVisibility attribute*), 270
ColorSpaceAliasIterator (*class in PyOpenColorIO.ColorSpace*), 201
ColorSpaceCategoryIterator (*class in PyOpenColorIO.ColorSpace*), 201
ColorSpaceDirection (*class in PyOpenColorIO*), 271
ColorSpaceIterator (*class in PyOpenColorIO.ColorSpaceSet*), 208
ColorSpaceIterator (*class in PyOpenColorIO.Config*), 229
ColorSpaceLevelIterator (*class in PyOpenColorIO.ColorSpaceMenuHelper*), 181
ColorSpaceMenuHelper (*class in PyOpenColorIO*), 180

ColorSpaceMenuParameters (*class in PyOpenColorIO*), 178
ColorSpaceNameIterator (*class in PyOpenColorIO.ColorSpaceSet*), 208
ColorSpaceNameIterator (*class in PyOpenColorIO.Config*), 229
ColorSpaceSet (*class in PyOpenColorIO*), 206
ColorSpaceTransform (*class in PyOpenColorIO*), 397
ColorSpaceVisibility (*class in PyOpenColorIO*), 270
CombineTransformDirections() (*in module PyOpenColorIO*), 307
Config (*class in PyOpenColorIO*), 210
Config.ActiveDisplaysListIterator (*class in PyOpenColorIO*), 210
Config.ActiveViewsListIterator (*class in PyOpenColorIO*), 210
Config.DisplayAllIterator (*class in PyOpenColorIO*), 212
Config.ViewForViewTypeIterator (*class in PyOpenColorIO*), 215
Config.VirtualViewIterator (*class in PyOpenColorIO*), 215
ConstChildElementIterator (*class in PyOpenColorIO.FormatMetadata*), 301
Context (*class in PyOpenColorIO*), 256
contrast (*PyOpenColorIO.GradingPrimary property*), 314
CPUProcessor (*class in PyOpenColorIO*), 351
CreateFromBuiltinConfig() (*PyOpenColorIO.Config static method*), 211
CreateFromConfigIOProxy() (*PyOpenColorIO.Config static method*), 211
CreateFromEnv() (*PyOpenColorIO.Config static method*), 212
CreateFromFile() (*PyOpenColorIO.CDLTransform static method*), 393
CreateFromFile() (*PyOpenColorIO.Config static method*), 212
CreateFromStream() (*PyOpenColorIO.Config static method*), 212
CreateGroupFromFile() (*PyOpenColorIO.CDLTransform static method*), 394
createGroupTransform() (*PyOpenColorIO.Processor method*), 347
CreateRaw() (*PyOpenColorIO.Config static method*), 212
CreateShaderDesc() (*PyOpenColorIO.GpuShaderDesc static method*), 377
createShaderText() (*PyOpenColorIO.GpuShaderCreator method*), 368
createShaderText() (*PyOpenColorIO.GpuShaderDesc method*), 378

D

decreaseRulePriority() (*PyOpenColorIO.FileRules method*), 358
dimensions (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
DisplayIterator (*class in PyOpenColorIO.Config*), 230
DisplayViewTransform (*class in PyOpenColorIO*), 398
DYLD_LIBRARY_PATH, 21
DYNAMIC_PROPERTY_CONTRAST (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_EXPOSURE (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_GAMMA (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_GRADING_HUECURVE (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_GRADING_PRIMARY (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_GRADING_RGBCURVE (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DYNAMIC_PROPERTY_GRADING_TONE (*PyOpenColorIO.DynamicPropertyType attribute*), 291
DynamicProperty (*class in PyOpenColorIO*), 261
DynamicPropertyIterator (*class in PyOpenColorIO.GpuShaderCreator*), 369
DynamicPropertyIterator (*class in PyOpenColorIO.GpuShaderDesc*), 382
DynamicPropertyType (*class in PyOpenColorIO*), 291

E

edgeLen (*PyOpenColorIO.GpuShaderDesc.Texture3D property*), 381
end() (*PyOpenColorIO.GpuShaderCreator method*), 368
end() (*PyOpenColorIO.GpuShaderDesc method*), 379
ENV_ENVIRONMENT_LOAD_ALL (*PyOpenColorIO.EnvironmentMode attribute*), 282
ENV_ENVIRONMENT_LOAD_PREDEFINED (*PyOpenColorIO.EnvironmentMode attribute*), 282
ENV_ENVIRONMENT_UNKNOWN (*PyOpenColorIO.EnvironmentMode attribute*), 282
environment variable
    DYLD_LIBRARY_PATH, 21
    DYLD_LIBRARY_PATH (*macOS*), 21
    LD_LIBRARY_PATH, 21
    LD_LIBRARY_PATH (*Linux*), 21
    OCIO, 20, 21, 36, 94, 210
    OCIO_ACTIVE_DISPLAYS, 37
    OCIO_ACTIVE_VIEWS, 37
    OCIO_INACTIVE_COLORSPACES, 37, 57
    OCIO_LOGGING_LEVEL, 37
    OCIO_OPTIMIZATION_FLAGS, 37

OCIO_USER_CATEGORIES, 37
OFX_PLUGIN_PATH, 21
PYTHONPATH, 21
EnvironmentMode (*class in PyOpenColorIO*), 282
EnvironmentModeFromString() (*in module PyOpenColorIO*), 309
EnvironmentModeToString() (*in module PyOpenColorIO*), 309
EnvironmentVarNameIterator (*class in PyOpenColorIO.Config*), 229
equals() (*PyOpenColorIO.CDLTransform method*), 394
equals() (*PyOpenColorIO.ExponentTransform method*), 401
equals() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
equals() (*PyOpenColorIO.ExposureContrastTransform method*), 405
equals() (*PyOpenColorIO.FixedFunctionTransform method*), 411
equals() (*PyOpenColorIO.LogAffineTransform method*), 416
equals() (*PyOpenColorIO.LogCameraTransform method*), 418
equals() (*PyOpenColorIO.LogTransform method*), 421
equals() (*PyOpenColorIO.Lut1DTransform method*), 424
equals() (*PyOpenColorIO.Lut3DTransform method*), 427
equals() (*PyOpenColorIO.MatrixTransform method*), 430
equals() (*PyOpenColorIO.RangeTransform method*), 433
Exception, 299
ExceptionMissingFile, 299
ExponentTransform (*class in PyOpenColorIO*), 400
ExponentWithLinearTransform (*class in PyOpenColorIO*), 402
exposure (*PyOpenColorIO.GradingPrimary property*), 314
EXPOSURE_CONTRAST_LINEAR (*PyOpenColorIO.ExposureContrastStyle attribute*), 287
EXPOSURE_CONTRAST_LOGARITHMIC (*PyOpenColorIO.ExposureContrastStyle attribute*), 288
EXPOSURE_CONTRAST_VIDEO (*PyOpenColorIO.ExposureContrastStyle attribute*), 288
ExposureContrastStyle (*class in PyOpenColorIO*), 287
ExposureContrastStyleFromString() (*in module PyOpenColorIO*), 310
ExposureContrastStyleToString() (*in module PyOpenColorIO*), 310
ExposureContrastTransform (*class in PyOpenColorIO*), 405
extractGpuShaderInfo() (*PyOpenColorIO.GPUProcessor method*), 355

F

FileIterator (*class in PyOpenColorIO.ProcessorMetadata*), 356
filepathOnlyMatchesDefaultRule() (*PyOpenColorIO.Config method*), 217
FileRules (*class in PyOpenColorIO*), 357
FileTransform (*class in PyOpenColorIO*), 408
finalize() (*PyOpenColorIO.GpuShaderCreator method*), 368
finalize() (*PyOpenColorIO.GpuShaderDesc method*), 379
Fit() (*PyOpenColorIO.MatrixTransform static method*), 430
FIXED_FUNCTION_ACES_DARK_TO_DIM_10 (*PyOpenColorIO.FixedFunctionStyle attribute*), 284
FIXED_FUNCTION_ACES_GAMUT_COMP_13 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_GAMUT_COMPRESS_20 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_GAMUTMAP_02 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_GAMUTMAP_07 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_GLOW_03 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_GLOW_10 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_OUTPUT_TRANSFORM_20 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_RED_MOD_03 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_RED_MOD_10 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_RGB_TO_JMH_20 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_ACES_TONESCALE_COMPRESS_20 (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_LIN_TO_DOUBLE_LOG (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_LIN_TO_GAMMA_LOG (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_LIN_TO_PQ (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_REC2100_SURROUND (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_RGB_TO_HSV (*PyOpenColorIO.FixedFunctionStyle attribute*), 285

FIXED_FUNCTION_RGB_TO_HSY_LIN (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_RGB_TO_HSY_LOG (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_RGB_TO_HSY_VID (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_XYZ_TO_LUV (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_XYZ_TO_uvY (*PyOpenColorIO.FixedFunctionStyle attribute*), 285
FIXED_FUNCTION_XYZ_TO_xyY (*PyOpenColorIO.FixedFunctionStyle attribute*), 286
FixedFunctionStyle (*class in PyOpenColorIO*), 284
FixedFunctionStyleFromString() (*in module PyOpenColorIO*), 310
FixedFunctionStyleToString() (*in module PyOpenColorIO*), 310
FixedFunctionTransform (*class in PyOpenColorIO*), 411
FormatIterator (*class in PyOpenColorIO.Baker*), 196
FormatIterator (*class in PyOpenColorIO.FileTransform*), 409
FormatMetadata (*class in PyOpenColorIO*), 300

G

gain (*PyOpenColorIO.GradingPrimary property*), 314
gamma (*PyOpenColorIO.GradingPrimary property*), 314
get3DTextures() (*PyOpenColorIO.GpuShaderDesc method*), 379
getActiveDisplays() (*PyOpenColorIO.Config method*), 218
getActiveViews() (*PyOpenColorIO.Config method*), 218
getAData() (*PyOpenColorIO.PlanarImageDesc method*), 339
getAddedColorSpaces() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getAliases() (*PyOpenColorIO.ColorSpace method*), 199
getAliases() (*PyOpenColorIO.NamedTransform method*), 344
getAllocation() (*PyOpenColorIO.AllocationTransform method*), 390
getAllocation() (*PyOpenColorIO.ColorSpace method*), 199
getAllocationVars() (*PyOpenColorIO.ColorSpace method*), 199
getAllowTexture1D() (*PyOpenColorIO.GpuShaderCreator method*), 368
getAllowTexture1D() (*PyOpenColorIO.GpuShaderDesc method*), 379
getAppCategories() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getAttributes() (*PyOpenColorIO.FormatMetadata method*), 300
getBase() (*PyOpenColorIO.LogAffineTransform method*), 416
getBase() (*PyOpenColorIO.LogCameraTransform method*), 418
getBase() (*PyOpenColorIO.LogTransform method*), 421
getBData() (*PyOpenColorIO.PlanarImageDesc method*), 339
getBitDepth() (*PyOpenColorIO.ColorSpace method*), 199
getBitDepth() (*PyOpenColorIO.ImageDesc method*), 334
getBitDepth() (*PyOpenColorIO.PackedImageDesc method*), 337
getBitDepth() (*PyOpenColorIO.PlanarImageDesc method*), 339
getBool() (*PyOpenColorIO.GpuShaderDesc.UniformData method*), 380
getBuiltins() (*PyOpenColorIO.BuiltinTransformRegistry method*), 392
getBypassLinToLog() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
getCacheID() (*PyOpenColorIO.Config method*), 218
getCacheID() (*PyOpenColorIO.Context method*), 257
getCacheID() (*PyOpenColorIO.CPUProcessor method*), 353
getCacheID() (*PyOpenColorIO.GPUProcessor method*), 355
getCacheID() (*PyOpenColorIO.GpuShaderCreator method*), 368
getCacheID() (*PyOpenColorIO.GpuShaderDesc method*), 379
getCacheID() (*PyOpenColorIO.Processor method*), 347
getCanonicalName() (*PyOpenColorIO.Config method*), 218
getCategories() (*PyOpenColorIO.ColorSpace method*), 199
getCategories() (*PyOpenColorIO.NamedTransform method*), 344
getCategories() (*PyOpenColorIO.ViewTransform method*), 438
getCCCId() (*PyOpenColorIO.FileTransform method*), 409
getCDLStyle() (*PyOpenColorIO.FileTransform method*), 409
getChannelOrder() (*PyOpenColorIO.PackedImageDesc method*), 337

getChannelView() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getChanStrideBytes() (*PyOpenColorIO.PackedImageDesc method*), 337
getChildElements() (*PyOpenColorIO.FormatMetadata method*), 300
getColorSpace() (*PyOpenColorIO.ColorSpaceSet method*), 207
getColorSpace() (*PyOpenColorIO.Config method*), 218
getColorSpace() (*PyOpenColorIO.FileRules method*), 358
getColorSpaceFromFilepath() (*PyOpenColorIO.Config method*), 218
getColorSpaceNames() (*PyOpenColorIO.ColorSpaceSet method*), 207
getColorSpaceNames() (*PyOpenColorIO.Config method*), 218
getColorSpaces() (*PyOpenColorIO.ColorSpaceSet method*), 207
getColorSpaces() (*PyOpenColorIO.Config method*), 219
getColorSpaces() (*PyOpenColorIO.ViewingRules method*), 363
getColorTimingCC() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getConfig() (*PyOpenColorIO.Baker method*), 195
getConfig() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getConfigIOProxy() (*PyOpenColorIO.Config method*), 219
getContrast() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getControlPoints() (*PyOpenColorIO.GradingBSplineCurve method*), 328
getCubeSize() (*PyOpenColorIO.Baker method*), 195
getCurrentContext() (*PyOpenColorIO.Config method*), 219
getCustomKeyName() (*PyOpenColorIO.FileRules method*), 358
getCustomKeyName() (*PyOpenColorIO.ViewingRules method*), 363
getCustomKeyValue() (*PyOpenColorIO.FileRules method*), 358
getCustomKeyValue() (*PyOpenColorIO.ViewingRules method*), 363
getData() (*PyOpenColorIO.Lut1DTransform method*), 424
getData() (*PyOpenColorIO.Lut3DTransform method*), 427
getData() (*PyOpenColorIO.PackedImageDesc method*), 337
getDataBypass() (*PyOpenColorIO.ColorSpaceTransform method*), 397
getDataBypass() (*PyOpenColorIO.DisplayViewTransform method*), 399
getDefaultCPUProcessor() (*PyOpenColorIO.Processor method*), 347
getDefaultDisplay() (*PyOpenColorIO.Config method*), 219
getDefaultGPUProcessor() (*PyOpenColorIO.Processor method*), 347
getDefaultLumaCoefs() (*PyOpenColorIO.Config method*), 219
getDefaultSceneToDisplayViewTransform() (*PyOpenColorIO.Config method*), 219
getDefaultView() (*PyOpenColorIO.Config method*), 219
getDefaultViewTransformName() (*PyOpenColorIO.Config method*), 220
getDescription() (*PyOpenColorIO.BuiltinTransform method*), 391
getDescription() (*PyOpenColorIO.ColorSpace method*), 199
getDescription() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getDescription() (*PyOpenColorIO.Config method*), 220
getDescription() (*PyOpenColorIO.Look method*), 341
getDescription() (*PyOpenColorIO.NamedTransform method*), 344
getDescription() (*PyOpenColorIO.ViewTransform method*), 438
getDescriptorSetIndex() (*PyOpenColorIO.GpuShaderCreator method*), 368
getDescriptorSetIndex() (*PyOpenColorIO.GpuShaderDesc method*), 379
getDirection() (*PyOpenColorIO.AllocationTransform method*), 390
getDirection() (*PyOpenColorIO.BuiltinTransform method*), 391
getDirection() (*PyOpenColorIO.CDLTransform method*), 394
getDirection() (*PyOpenColorIO.ColorSpaceTransform method*), 397
getDirection() (*PyOpenColorIO.DisplayViewTransform method*), 399
getDirection() (*PyOpenColorIO.ExponentTransform method*), 401
getDirection() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getDirection() (*PyOpenColorIO.ExposureContrastTransform method*), 405

getDirection() (*PyOpenColorIO.FileTransform method*), 409
getDirection() (*PyOpenColorIO.FixedFunctionTransform method*), 411
getDirection() (*PyOpenColorIO.GradingHueCurveTransform method*), 321
getDirection() (*PyOpenColorIO.GradingPrimaryTransform method*), 311
getDirection() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
getDirection() (*PyOpenColorIO.GradingToneTransform method*), 330
getDirection() (*PyOpenColorIO.GroupTransform method*), 413
getDirection() (*PyOpenColorIO.LogAffineTransform method*), 416
getDirection() (*PyOpenColorIO.LogCameraTransform method*), 418
getDirection() (*PyOpenColorIO.LogTransform method*), 421
getDirection() (*PyOpenColorIO.LookTransform method*), 422
getDirection() (*PyOpenColorIO.Lut1DTransform method*), 424
getDirection() (*PyOpenColorIO.Lut3DTransform method*), 427
getDirection() (*PyOpenColorIO.MatrixTransform method*), 430
getDirection() (*PyOpenColorIO.RangeTransform method*), 433
getDirection() (*PyOpenColorIO.Transform method*), 388
getDisplay() (*PyOpenColorIO.Baker method*), 195
getDisplay() (*PyOpenColorIO.DisplayViewTransform method*), 399
getDisplayCC() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getDisplays() (*PyOpenColorIO.Config method*), 220
getDisplaysAll() (*PyOpenColorIO.Config method*), 220
getDisplayViewColorSpaceName() (*PyOpenColorIO.Config method*), 220
getDisplayViewDescription() (*PyOpenColorIO.Config method*), 220
getDisplayViewLooks() (*PyOpenColorIO.Config method*), 220
getDisplayViewRule() (*PyOpenColorIO.Config method*), 220
getDisplayViewTransform() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getDisplayViewTransformName() (*PyOpenColorIO.Config method*), 220
getDouble() (*PyOpenColorIO.DynamicProperty method*), 262
getDouble() (*PyOpenColorIO.GpuShaderDesc.UniformData method*), 381
getDrawCurveOnly() (*PyOpenColorIO.GradingHueCurve method*), 325
getDst() (*PyOpenColorIO.ColorSpaceTransform method*), 397
getDst() (*PyOpenColorIO.LookTransform method*), 422
getDynamicProperties() (*PyOpenColorIO.GpuShaderCreator method*), 368
getDynamicProperties() (*PyOpenColorIO.GpuShaderDesc method*), 379
getDynamicProperty() (*PyOpenColorIO.CPUProcessor method*), 353
getDynamicProperty() (*PyOpenColorIO.GpuShaderCreator method*), 368
getDynamicProperty() (*PyOpenColorIO.GpuShaderDesc method*), 379
getDynamicProperty() (*PyOpenColorIO.Processor method*), 347
getElementName() (*PyOpenColorIO.FormatMetadata method*), 300
getElementValue() (*PyOpenColorIO.FormatMetadata method*), 301
getEncoding() (*PyOpenColorIO.ColorSpace method*), 199
getEncoding() (*PyOpenColorIO.NamedTransform method*), 344
getEncodings() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getEncodings() (*PyOpenColorIO.ViewingRules method*), 363
getEnvironmentMode() (*PyOpenColorIO.Config method*), 220
getEnvironmentMode() (*PyOpenColorIO.Context method*), 257
getEnvironmentVarDefault() (*PyOpenColorIO.Config method*), 220
getEnvironmentVarNames() (*PyOpenColorIO.Config method*), 220
GetEnvVariable() (*in module PyOpenColorIO*), 306
getEqualityGroup() (*PyOpenColorIO.ColorSpace method*), 199
getExposure() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getExtension() (*PyOpenColorIO.FileRules method*), 358
getFamily() (*PyOpenColorIO.ColorSpace method*),

199
getFamily() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getFamily() (*PyOpenColorIO.NamedTransform method*), 344
getFamily() (*PyOpenColorIO.ViewTransform method*), 438
getFamilySeparator() (*PyOpenColorIO.Config method*), 220
getFileInputBitDepth() (*PyOpenColorIO.MatrixTransform method*), 430
getFileInputBitDepth() (*PyOpenColorIO.RangeTransform method*), 433
getFileOutputBitDepth() (*PyOpenColorIO.Lut1DTransform method*), 424
getFileOutputBitDepth() (*PyOpenColorIO.Lut3DTransform method*), 428
getFileOutputBitDepth() (*PyOpenColorIO.MatrixTransform method*), 431
getFileOutputBitDepth() (*PyOpenColorIO.RangeTransform method*), 434
getFileRules() (*PyOpenColorIO.Config method*), 220
getFiles() (*PyOpenColorIO.ProcessorMetadata method*), 356
getFirstSOPDescription() (*PyOpenColorIO.CDLTransform method*), 394
getFloat3() (*PyOpenColorIO.GpuShaderDesc.UniformData method*), 381
getFormat() (*PyOpenColorIO.Baker method*), 195
getFormatMetadata() (*PyOpenColorIO.Baker method*), 195
getFormatMetadata() (*PyOpenColorIO.CDLTransform method*), 394
getFormatMetadata() (*PyOpenColorIO.ExponentTransform method*), 401
getFormatMetadata() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getFormatMetadata() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getFormatMetadata() (*PyOpenColorIO.FixedFunctionTransform method*), 411
getFormatMetadata() (*PyOpenColorIO.GradingHueCurveTransform method*), 321
getFormatMetadata() (*PyOpenColorIO.GradingPrimaryTransform method*), 311
getFormatMetadata() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
getFormatMetadata() (*PyOpenColorIO.GradingToneTransform method*), 330
getFormatMetadata() (*PyOpenColorIO.GroupTransform method*), 413
getFormatMetadata() (*PyOpenColorIO.LogAffineTransform method*), 416
getFormatMetadata() (*PyOpenColorIO.LogCameraTransform method*), 418
getFormatMetadata() (*PyOpenColorIO.LogTransform method*), 421
getFormatMetadata() (*PyOpenColorIO.Lut1DTransform method*), 424
getFormatMetadata() (*PyOpenColorIO.Lut3DTransform method*), 428
getFormatMetadata() (*PyOpenColorIO.MatrixTransform method*), 431
getFormatMetadata() (*PyOpenColorIO.Processor method*), 348
getFormatMetadata() (*PyOpenColorIO.RangeTransform method*), 434
getFormats() (*PyOpenColorIO.Baker static method*), 195
getFormats() (*PyOpenColorIO.FileTransform static method*), 409
getFunctionName() (*PyOpenColorIO.GpuShaderCreator method*), 368
getFunctionName() (*PyOpenColorIO.GpuShaderDesc method*), 379
getGamma() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getGamma() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getGData() (*PyOpenColorIO.PlanarImageDesc method*), 339
getGradingHueCurve() (*PyOpenColorIO.DynamicProperty method*), 262
getGradingPrimary() (*PyOpenColorIO.DynamicProperty method*), 262
getGradingRGBCurve() (*PyOpenColorIO.DynamicProperty method*), 262
getGradingTone() (*PyOpenColorIO.DynamicProperty method*), 262
getGridSize() (*PyOpenColorIO.Lut3DTransform method*), 428
getHeight() (*PyOpenColorIO.ImageDesc method*), 334
getHeight() (*PyOpenColorIO.PackedImageDesc method*), 337
getHeight() (*PyOpenColorIO.PlanarImageDesc method*), 339
getHierarchyLevels() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getHueAdjust() (*PyOpenColorIO.Lut1DTransform*

method), 424
getID() (*PyOpenColorIO.CDLTransform method*), 394
getID() (*PyOpenColorIO.FormatMetadata method*), 301
GetIdentityProcessor() (*in module PyOpenColorIO.DisplayViewHelpers*), 186
getInactiveColorSpaces() (*PyOpenColorIO.Config method*), 220
getIncludeColorSpaces() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getIncludeNamedTransforms() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getIncludeRoles() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 179
getIndexForRule() (*PyOpenColorIO.FileRules method*), 358
getIndexForRule() (*PyOpenColorIO.ViewingRules method*), 363
getIndexFromName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getIndexFromUIName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getInputBitDepth() (*PyOpenColorIO.CPUProcessor method*), 353
getInputHalfDomain() (*PyOpenColorIO.Lut1DTransform method*), 424
getInputSpace() (*PyOpenColorIO.Baker method*), 195
getInterchangeAttribute() (*PyOpenColorIO.ColorSpace method*), 199
getInterchangeAttribute() (*PyOpenColorIO.Look method*), 341
getInterchangeAttribute() (*PyOpenColorIO.ViewTransform method*), 438
getInterchangeAttributes() (*PyOpenColorIO.ColorSpace method*), 200
getInterchangeAttributes() (*PyOpenColorIO.Look method*), 341
getInterchangeAttributes() (*PyOpenColorIO.ViewTransform method*), 438
getInteropID() (*PyOpenColorIO.ColorSpace method*), 200
getInterpolation() (*PyOpenColorIO.FileTransform method*), 409
getInterpolation() (*PyOpenColorIO.Lut1DTransform method*), 424
getInterpolation() (*PyOpenColorIO.Lut3DTransform method*), 428
getInverseTransform() (*PyOpenColorIO.Look method*), 341
GetInverseTransformDirection() (*in module PyOpenColorIO*), 307
getLanguage() (*PyOpenColorIO.GpuShaderCreator method*), 368
getLanguage() (*PyOpenColorIO.GpuShaderDesc method*), 379
getLength() (*PyOpenColorIO.Lut1DTransform method*), 424
getLinearCC() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getLinearSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLinSideBreakValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLinSideOffsetValue() (*PyOpenColorIO.LogAffineTransform method*), 416
getLinSideOffsetValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLinSideSlopeValue() (*PyOpenColorIO.LogAffineTransform method*), 416
getLinSideSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLogExposureStep() (*PyOpenColorIO.ExposureContrastTransform method*), 405
GetLoggingLevel() (*in module PyOpenColorIO*), 304
getLogMidGray() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getLogSideOffsetValue() (*PyOpenColorIO.LogAffineTransform method*), 416
getLogSideOffsetValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLogSideSlopeValue() (*PyOpenColorIO.LogAffineTransform method*), 416
getLogSideSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 418
getLook() (*PyOpenColorIO.Config method*), 220
getLookNames() (*PyOpenColorIO.Config method*), 220
getLooks() (*PyOpenColorIO.Baker method*), 195
getLooks() (*PyOpenColorIO.Config method*), 220
getLooks() (*PyOpenColorIO.LookTransform method*), 422
getLooks() (*PyOpenColorIO.ProcessorMetadata method*), 356
getLooksBypass() (*PyOpenColorIO.DisplayViewTransform method*), 399
getLooksOverride() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getLooksOverrideEnabled() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getMajorVersion() (*PyOpenColorIO.Config method*), 220
getMatrix() (*PyOpenColorIO.MatrixTransform method*), 431

getMaxInValue() (*PyOpenColorIO.RangeTransform method*), 434
getMaxOutValue() (*PyOpenColorIO.RangeTransform method*), 434
getMinInValue() (*PyOpenColorIO.RangeTransform method*), 434
getMinorVersion() (*PyOpenColorIO.Config method*), 220
getMinOutValue() (*PyOpenColorIO.RangeTransform method*), 434
getMixingEncodings() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
getMixingSpaces() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
getMonitors() (*PyOpenColorIO.SystemMonitors method*), 387
getName() (*PyOpenColorIO.ColorSpace method*), 200
getName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getName() (*PyOpenColorIO.Config method*), 220
getName() (*PyOpenColorIO.FileRules method*), 358
getName() (*PyOpenColorIO.FormatMetadata method*), 301
getName() (*PyOpenColorIO.Look method*), 341
getName() (*PyOpenColorIO.NamedTransform method*), 344
getName() (*PyOpenColorIO.ViewingRules method*), 363
getName() (*PyOpenColorIO.ViewTransform method*), 438
getNamedTransform() (*PyOpenColorIO.Config method*), 221
getNamedTransformNames() (*PyOpenColorIO.Config method*), 221
getNamedTransforms() (*PyOpenColorIO.Config method*), 221
getNameFromUIName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getNegativeStyle() (*PyOpenColorIO.ExponentTransform method*), 401
getNegativeStyle() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getNextResourceIndex() (*PyOpenColorIO.GpuShaderCreator method*), 368
getNextResourceIndex() (*PyOpenColorIO.GpuShaderDesc method*), 379
getNumActiveDisplays() (*PyOpenColorIO.Config method*), 221
getNumActiveViews() (*PyOpenColorIO.Config method*), 221
getNumChannels() (*PyOpenColorIO.PackedImageDesc method*), 337
getNumColorSpaces() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getNumCustomKeys() (*PyOpenColorIO.FileRules method*), 358
getNumCustomKeys() (*PyOpenColorIO.ViewingRules method*), 363
getNumEntries() (*PyOpenColorIO.FileRules method*), 358
getNumEntries() (*PyOpenColorIO.ViewingRules method*), 363
getOffset() (*PyOpenColorIO.CDLTransform method*), 394
getOffset() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getOffset() (*PyOpenColorIO.MatrixTransform method*), 431
getOptimizedCPUProcessor() (*PyOpenColorIO.Processor method*), 348
getOptimizedGPUProcessor() (*PyOpenColorIO.Processor method*), 348
getOptimizedProcessor() (*PyOpenColorIO.Processor method*), 348
getOutputBitDepth() (*PyOpenColorIO.CPUProcessor method*), 353
getOutputRawHalfs() (*PyOpenColorIO.Lut1DTransform method*), 424
getParams() (*PyOpenColorIO.FixedFunctionTransform method*), 411
getPattern() (*PyOpenColorIO.FileRules method*), 358
getPivot() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getPixelName() (*PyOpenColorIO.GpuShaderCreator method*), 368
getPixelName() (*PyOpenColorIO.GpuShaderDesc method*), 379
getPower() (*PyOpenColorIO.CDLTransform method*), 394
GetProcessor() (*in module PyOpenColorIO.DisplayViewHelpers*), 186
getProcessor() (*PyOpenColorIO.Config method*), 221
getProcessor() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
getProcessor() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
GetProcessorFromBuiltinColorSpace() (*PyOpenColorIO.Config static method*), 212
GetProcessorFromConfigs() (*PyOpenColorIO.Config static method*), 213
getProcessorMetadata() (*PyOpenColorIO.Processor method*), 348
GetProcessorToBuiltinColorSpace() (*PyOpenCol-*

orIO.Config static method), 214
getProcessSpace() (PyOpenColorIO.Look method), 341
getRData() (PyOpenColorIO.PlanarImageDesc method), 340
getReferenceSpaceType() (PyOpenColorIO.ColorSpace method), 200
getReferenceSpaceType() (PyOpenColorIO.ViewTransform method), 438
getRegex() (PyOpenColorIO.FileRules method), 358
getResourcePrefix() (PyOpenColorIO.GpuShaderCreator method), 368
getResourcePrefix() (PyOpenColorIO.GpuShaderDesc method), 379
getRGBToHSY() (PyOpenColorIO.GradingHueCurveTransform method), 321
getRole() (PyOpenColorIO.ColorSpaceMenuParameters method), 179
getRoleColorSpace() (PyOpenColorIO.Config method), 222
getRoleNames() (PyOpenColorIO.Config method), 222
getRoles() (PyOpenColorIO.Config method), 222
getSat() (PyOpenColorIO.CDLTransform method), 394
getSatLumaCoefs() (PyOpenColorIO.CDLTransform method), 394
getSearchPath() (PyOpenColorIO.Config method), 222
getSearchPath() (PyOpenColorIO.Context method), 257
getSearchPaths() (PyOpenColorIO.Config method), 222
getSearchPaths() (PyOpenColorIO.Context method), 257
getSearchReferenceSpaceType() (PyOpenColorIO.ColorSpaceMenuParameters method), 180
getSelectedMixingEncodingIdx() (PyOpenColorIO.MixingColorSpaceManager method), 191
getSelectedMixingSpaceIdx() (PyOpenColorIO.MixingColorSpaceManager method), 191
getShaderText() (PyOpenColorIO.GpuShaderDesc method), 379
getShaperSize() (PyOpenColorIO.Baker method), 195
getShaperSpace() (PyOpenColorIO.Baker method), 195
getSharedViews() (PyOpenColorIO.Config method), 222
getSkipColorSpaceConversion() (PyOpenColorIO.LookTransform method), 422
getSlider() (PyOpenColorIO.MixingColorSpaceManager method), 191
getSliderMaxEdge() (PyOpenColorIO.MixingSlider method), 191
getSliderMinEdge() (PyOpenColorIO.MixingSlider method), 191
getSlope() (PyOpenColorIO.CDLTransform method), 394
getSlope() (PyOpenColorIO.GradingHueCurveTransform method), 322
getSlope() (PyOpenColorIO.GradingRGBCurveTransform method), 317
getSlopes() (PyOpenColorIO.GradingBSplineCurve method), 328
getSOP() (PyOpenColorIO.CDLTransform method), 394
getSplineType() (PyOpenColorIO.GradingBSplineCurve method), 328
getSrc() (PyOpenColorIO.ColorSpaceTransform method), 397
getSrc() (PyOpenColorIO.DisplayViewTransform method), 399
getSrc() (PyOpenColorIO.FileTransform method), 409
getSrc() (PyOpenColorIO.LookTransform method), 422
getStringVars() (PyOpenColorIO.Context method), 257
getStyle() (PyOpenColorIO.BuiltinTransform method), 391
getStyle() (PyOpenColorIO.CDLTransform method), 394
getStyle() (PyOpenColorIO.ExposureContrastTransform method), 405
getStyle() (PyOpenColorIO.FixedFunctionTransform method), 411
getStyle() (PyOpenColorIO.GradingHueCurveTransform method), 322
getStyle() (PyOpenColorIO.GradingPrimaryTransform method), 312
getStyle() (PyOpenColorIO.GradingRGBCurveTransform method), 317
getStyle() (PyOpenColorIO.GradingToneTransform method), 330
getStyle() (PyOpenColorIO.RangeTransform method), 434
getTargetSpace() (PyOpenColorIO.Baker method),

195
getTextureBindingStart() (*PyOpenColorIO.GpuShaderCreator method*), 368
getTextureBindingStart() (*PyOpenColorIO.GpuShaderDesc method*), 379
getTextureMaxWidth() (*PyOpenColorIO.GpuShaderCreator method*), 368
getTextureMaxWidth() (*PyOpenColorIO.GpuShaderDesc method*), 379
getTextures() (*PyOpenColorIO.GpuShaderDesc method*), 379
getTransform() (*PyOpenColorIO.ColorSpace method*), 200
getTransform() (*PyOpenColorIO.Look method*), 341
getTransform() (*PyOpenColorIO.NamedTransform method*), 344
GetTransform() (*PyOpenColorIO.NamedTransform static method*), 343
getTransform() (*PyOpenColorIO.ViewTransform method*), 438
getTransformFormatMetadata() (*PyOpenColorIO.Processor method*), 348
getTransformType() (*PyOpenColorIO.AllocationTransform method*), 390
getTransformType() (*PyOpenColorIO.BuiltinTransform method*), 391
getTransformType() (*PyOpenColorIO.CDLTransform method*), 395
getTransformType() (*PyOpenColorIO.ColorSpaceTransform method*), 397
getTransformType() (*PyOpenColorIO.DisplayViewTransform method*), 399
getTransformType() (*PyOpenColorIO.ExponentTransform method*), 401
getTransformType() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
getTransformType() (*PyOpenColorIO.ExposureContrastTransform method*), 405
getTransformType() (*PyOpenColorIO.FileTransform method*), 409
getTransformType() (*PyOpenColorIO.FixedFunctionTransform method*), 411
getTransformType() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
getTransformType() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
getTransformType() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
getTransformType() (*PyOpenColorIO.GradingToneTransform method*), 330
getTransformType() (*PyOpenColorIO.GroupTransform method*), 413
getTransformType() (*PyOpenColorIO.LogAffineTransform method*), 416
getTransformType() (*PyOpenColorIO.LogCameraTransform method*), 418
getTransformType() (*PyOpenColorIO.LogTransform method*), 421
getTransformType() (*PyOpenColorIO.LookTransform method*), 422
getTransformType() (*PyOpenColorIO.Lut1DTransform method*), 424
getTransformType() (*PyOpenColorIO.Lut3DTransform method*), 428
getTransformType() (*PyOpenColorIO.MatrixTransform method*), 431
getTransformType() (*PyOpenColorIO.RangeTransform method*), 434
getTransformType() (*PyOpenColorIO.Transform method*), 388
getTreatNoCategoryAsAny() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
getType() (*PyOpenColorIO.DynamicProperty method*), 263
getUIName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getUINameFromName() (*PyOpenColorIO.ColorSpaceMenuHelper method*), 181
getUniformBufferSize() (*PyOpenColorIO.GpuShaderDesc method*), 379
getUniforms() (*PyOpenColorIO.GpuShaderDesc method*), 379
getUniqueID() (*PyOpenColorIO.GpuShaderCreator method*), 369
getUniqueID() (*PyOpenColorIO.GpuShaderDesc method*), 380
getUserCategories() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
getValue() (*PyOpenColorIO.ExponentTransform method*), 401
getValue() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
getValue() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
getValue() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
getValue() (*PyOpenColorIO.GradingToneTransform method*), 330

getValue() (*PyOpenColorIO.Lut1DTransform method*), 425
getValue() (*PyOpenColorIO.Lut3DTransform method*), 428
getValues() (*PyOpenColorIO.GpuShaderDesc.Texture method*), 381
getValues() (*PyOpenColorIO.GpuShaderDesc.Texture3D method*), 381
getVars() (*PyOpenColorIO.AllocationTransform method*), 390
getVectorFloat() (*PyOpenColorIO.GpuShaderDesc.UniformData method*), 381
getVectorInt() (*PyOpenColorIO.GpuShaderDesc.UniformData method*), 381
GetVersion() (*in module PyOpenColorIO*), 304
GetVersionHex() (*in module PyOpenColorIO*), 304
getView() (*PyOpenColorIO.Baker method*), 195
getView() (*PyOpenColorIO.DisplayViewTransform method*), 399
getViewingRules() (*PyOpenColorIO.Config method*), 223
getViews() (*PyOpenColorIO.Config method*), 223
getViewTransform() (*PyOpenColorIO.Config method*), 222
getViewTransformNames() (*PyOpenColorIO.Config method*), 223
getViewTransforms() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViewColorSpaceName() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViewDescription() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViewLooks() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViewRule() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViews() (*PyOpenColorIO.Config method*), 223
getVirtualDisplayViewTransformName() (*PyOpenColorIO.Config method*), 223
getWidth() (*PyOpenColorIO.ImageDesc method*), 334
getWidth() (*PyOpenColorIO.PackedImageDesc method*), 337
getWidth() (*PyOpenColorIO.PlanarImageDesc method*), 340
getWorkingDir() (*PyOpenColorIO.Config method*), 223
getWorkingDir() (*PyOpenColorIO.Context method*), 257
GetWriteFormats() (*PyOpenColorIO.GroupTransform static method*), 412
getXStrideBytes() (*PyOpenColorIO.ImageDesc method*), 335
getXStrideBytes() (*PyOpenColorIO.PackedImageDesc method*), 337
getXStrideBytes() (*PyOpenColorIO.PlanarImageDesc method*), 340
getYStrideBytes() (*PyOpenColorIO.ImageDesc method*), 335
getYStrideBytes() (*PyOpenColorIO.PackedImageDesc method*), 337
getYStrideBytes() (*PyOpenColorIO.PlanarImageDesc method*), 340
GPU_LANGUAGE_CG (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_1_2 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_1_3 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_4_0 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_ES_1_0 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_ES_3_0 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_GLSL_VK_4_6 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_HLSL_DX11 (*PyOpenColorIO.GpuLanguage attribute*), 281
GPU_LANGUAGE_MSL_2_0 (*PyOpenColorIO.GpuLanguage attribute*), 281
GpuLanguage (*class in PyOpenColorIO*), 281
GpuLanguageFromString() (*in module PyOpenColorIO*), 309
GpuLanguageToString() (*in module PyOpenColorIO*), 309
GPUProcessor (*class in PyOpenColorIO*), 355
GpuShaderCreator (*class in PyOpenColorIO*), 366
GpuShaderCreator.TextureDimensions (*class in PyOpenColorIO*), 367
GpuShaderDesc (*class in PyOpenColorIO*), 375
GpuShaderDesc.TextureDimensions (*class in PyOpenColorIO*), 377
GRADING_LIN (*PyOpenColorIO.GradingStyle attribute*), 290
GRADING_LOG (*PyOpenColorIO.GradingStyle attribute*), 290
GRADING_VIDEO (*PyOpenColorIO.GradingStyle attribute*), 290
GradingBSplineCurve (*class in PyOpenColorIO*), 327
GradingControlPoint (*class in PyOpenColorIO*), 326
GradingControlPointIterator (*class in PyOpenColorIO.GradingBSplineCurve*), 328
GradingHueCurve (*class in PyOpenColorIO*), 324
GradingHueCurveTransform (*class in PyOpenCol-*

orIO), 320
GradingPrimary (*class in PyOpenColorIO*), 313
GradingPrimaryTransform (*class in PyOpenColorIO*), 311
GradingRGBCurve (*class in PyOpenColorIO*), 319
GradingRGBCurveTransform (*class in PyOpenColorIO*), 316
GradingRGBM (*class in PyOpenColorIO*), 315
GradingRGBMSW (*class in PyOpenColorIO*), 333
GradingStyle (*class in PyOpenColorIO*), 290
GradingStyleFromString() (*in module PyOpenColorIO*), 310
GradingStyleToString() (*in module PyOpenColorIO*), 310
GradingTone (*class in PyOpenColorIO*), 332
GradingToneTransform (*class in PyOpenColorIO*), 329
green (*PyOpenColorIO.GradingRGBCurve property*), 319
green (*PyOpenColorIO.GradingRGBM property*), 315
green (*PyOpenColorIO.GradingRGBMSW property*), 333
GroupTransform (*class in PyOpenColorIO*), 412

H

hasAlias() (*PyOpenColorIO.ColorSpace method*), 200
hasAlias() (*PyOpenColorIO.NamedTransform method*), 344
hasCategory() (*PyOpenColorIO.ColorSpace method*), 200
hasCategory() (*PyOpenColorIO.NamedTransform method*), 344
hasCategory() (*PyOpenColorIO.ViewTransform method*), 438
hasChannelCrosstalk() (*PyOpenColorIO.CPUProcessor method*), 353
hasChannelCrosstalk() (*PyOpenColorIO.GPUProcessor method*), 355
hasChannelCrosstalk() (*PyOpenColorIO.Processor method*), 348
hasColorSpace() (*PyOpenColorIO.ColorSpaceSet method*), 207
hasDynamicProperty() (*PyOpenColorIO.CPUProcessor method*), 353
hasDynamicProperty() (*PyOpenColorIO.GpuShaderCreator method*), 369
hasDynamicProperty() (*PyOpenColorIO.GpuShaderDesc method*), 380
hasDynamicProperty() (*PyOpenColorIO.Processor method*), 348
hasMaxInValue() (*PyOpenColorIO.RangeTransform method*), 434
hasMaxOutValue() (*PyOpenColorIO.RangeTransform method*), 434
hasMinInValue() (*PyOpenColorIO.RangeTransform method*), 434
hasMinOutValue() (*PyOpenColorIO.RangeTransform method*), 434
hasRole() (*PyOpenColorIO.Config method*), 223
hasView() (*PyOpenColorIO.Config method*), 223
hasVirtualView() (*PyOpenColorIO.Config method*), 223
height (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
highlights (*PyOpenColorIO.GradingTone property*), 332
HUE_DW3 (*PyOpenColorIO.Lut1DHueAdjust attribute*), 278
hue_fx (*PyOpenColorIO.GradingHueCurve property*), 325
hue_hue (*PyOpenColorIO.GradingHueCurve property*), 325
hue_lum (*PyOpenColorIO.GradingHueCurve property*), 325
HUE_NONE (*PyOpenColorIO.Lut1DHueAdjust attribute*), 279
hue_sat (*PyOpenColorIO.GradingHueCurve property*), 325
HUE_WYPN (*PyOpenColorIO.Lut1DHueAdjust attribute*), 279

I

IdentifyBuiltinColorSpace() (*PyOpenColorIO.Config static method*), 214
IdentifyInterchangeSpace() (*PyOpenColorIO.Config static method*), 214
Identity() (*PyOpenColorIO.MatrixTransform static method*), 430
ImageDesc (*class in PyOpenColorIO*), 334
increaseRulePriority() (*PyOpenColorIO.FileRules method*), 358
insertPathSearchRule() (*PyOpenColorIO.FileRules method*), 358
insertRule() (*PyOpenColorIO.FileRules method*), 358
insertRule() (*PyOpenColorIO.ViewingRules method*), 363
instantiateDisplayFromICCProfile() (*PyOpenColorIO.Config method*), 224
instantiateDisplayFromMonitorName() (*PyOpenColorIO.Config method*), 224
INTERP_BEST (*PyOpenColorIO.Interpolation attribute*), 276
INTERP_CUBIC (*PyOpenColorIO.Interpolation attribute*), 276
INTERP_DEFAULT (*PyOpenColorIO.Interpolation attribute*), 276
INTERP_LINEAR (*PyOpenColorIO.Interpolation attribute*), 276

INTERP_NEAREST (*PyOpenColorIO.Interpolation attribute*), 276
INTERP_TETRAHEDRAL (*PyOpenColorIO.Interpolation attribute*), 276
INTERP_UNKNOWN (*PyOpenColorIO.Interpolation attribute*), 276
Interpolation (*class in PyOpenColorIO*), 275
interpolation (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
interpolation (*PyOpenColorIO.GpuShaderDesc.Texture3D property*), 381
InterpolationFromString() (*in module PyOpenColorIO*), 309
InterpolationToString() (*in module PyOpenColorIO*), 309
isArchivable() (*PyOpenColorIO.Config method*), 224
isColorSpaceLinear() (*PyOpenColorIO.Config method*), 224
isColorSpaceUsed() (*PyOpenColorIO.Config method*), 225
isContrastDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 405
isData() (*PyOpenColorIO.ColorSpace method*), 200
isDefault() (*PyOpenColorIO.FileRules method*), 359
isDisplayTemporary() (*PyOpenColorIO.Config method*), 225
isDynamic() (*PyOpenColorIO.CPUProcessor method*), 353
isDynamic() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
isDynamic() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
isDynamic() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
isDynamic() (*PyOpenColorIO.GradingToneTransform method*), 330
isDynamic() (*PyOpenColorIO.Processor method*), 348
IsEnvVariablePresent() (*in module PyOpenColorIO*), 306
isExposureDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 405
isFloat() (*PyOpenColorIO.ImageDesc method*), 335
isFloat() (*PyOpenColorIO.PackedImageDesc method*), 337
isFloat() (*PyOpenColorIO.PlanarImageDesc method*), 340
IsFormatExtensionSupported() (*PyOpenColorIO.FileTransform static method*), 408
isGammaDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 405
isIdentity() (*PyOpenColorIO.CPUProcessor method*), 353
isIdentity() (*PyOpenColorIO.GradingHueCurve method*), 325
isIdentity() (*PyOpenColorIO.GradingRGBCurve method*), 319
isInactiveColorSpace() (*PyOpenColorIO.Config method*), 225
isLinearSlopeValueSet() (*PyOpenColorIO.LogCameraTransform method*), 418
isNoOp() (*PyOpenColorIO.CPUProcessor method*), 353
isNoOp() (*PyOpenColorIO.GPUProcessor method*), 355
isNoOp() (*PyOpenColorIO.Processor method*), 349
isPerceptuallyUniform() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
isRGBAPacked() (*PyOpenColorIO.ImageDesc method*), 335
isRGBAPacked() (*PyOpenColorIO.PackedImageDesc method*), 337
isRGBAPacked() (*PyOpenColorIO.PlanarImageDesc method*), 340
isStrictParsingEnabled() (*PyOpenColorIO.Config method*), 225
isViewShared() (*PyOpenColorIO.Config method*), 225
isVirtualViewShared() (*PyOpenColorIO.Config method*), 226

L

LANGUAGE_OSL_1 (*PyOpenColorIO.GpuLanguage attribute*), 281
LD_LIBRARY_PATH, 21
LegacyViewingPipeline (*class in PyOpenColorIO*), 187
lift (*PyOpenColorIO.GradingPrimary property*), 314
loadEnvironment() (*PyOpenColorIO.Config method*), 226
loadEnvironment() (*PyOpenColorIO.Context method*), 257
LogAffineTransform (*class in PyOpenColorIO*), 415
LogCameraTransform (*class in PyOpenColorIO*), 418
LOGGING_LEVEL_DEBUG (*PyOpenColorIO.LoggingLevel attribute*), 267
LOGGING_LEVEL_INFO (*PyOpenColorIO.LoggingLevel attribute*), 267
LOGGING_LEVEL_NONE (*PyOpenColorIO.LoggingLevel attribute*), 267

LOGGING_LEVEL_UNKNOWN (*PyOpenColorIO.LoggingLevel attribute*), 267
LOGGING_LEVEL_WARNING (*PyOpenColorIO.LoggingLevel attribute*), 267
LoggingLevel (*class in PyOpenColorIO*), 267
LoggingLevelFromString() (*in module PyOpenColorIO*), 307
LoggingLevelToString() (*in module PyOpenColorIO*), 307
LogMessage() (*in module PyOpenColorIO*), 305
LogTransform (*class in PyOpenColorIO*), 420
Look (*class in PyOpenColorIO*), 341
LookIterator (*class in PyOpenColorIO.Config*), 231
LookIterator (*class in PyOpenColorIO.ProcessorMetadata*), 356
LookNameIterator (*class in PyOpenColorIO.Config*), 231
LookTransform (*class in PyOpenColorIO*), 422
lum_lum (*PyOpenColorIO.GradingHueCurve property*), 325
lum_sat (*PyOpenColorIO.GradingHueCurve property*), 325
Lut1DHueAdjust (*class in PyOpenColorIO*), 278
Lut1DTransform (*class in PyOpenColorIO*), 424
Lut3DTransform (*class in PyOpenColorIO*), 427

M

makeContrastDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeContrastNonDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeDynamic() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
makeDynamic() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
makeDynamic() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
makeDynamic() (*PyOpenColorIO.GradingToneTransform method*), 330
makeExposureDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeExposureNonDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeGammaDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeGammaNonDynamic() (*PyOpenColorIO.ExposureContrastTransform method*), 406
makeNonDynamic() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
makeNonDynamic() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
makeNonDynamic() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
makeNonDynamic() (*PyOpenColorIO.GradingToneTransform method*), 330
master (*PyOpenColorIO.GradingRGBCurve property*), 319
master (*PyOpenColorIO.GradingRGBM property*), 315
master (*PyOpenColorIO.GradingRGBMSW property*), 333
MatrixTransform (*class in PyOpenColorIO*), 430
midtones (*PyOpenColorIO.GradingTone property*), 332
MixingColorSpaceManager (*class in PyOpenColorIO*), 191
MixingEncodingIterator (*class in PyOpenColorIO.MixingColorSpaceManager*), 192
MixingSlider (*class in PyOpenColorIO*), 190
MixingSpaceIterator (*class in PyOpenColorIO.MixingColorSpaceManager*), 192
mixingToSlider() (*PyOpenColorIO.MixingSlider method*), 191
MonitorIterator (*class in PyOpenColorIO.SystemMonitors*), 387

N

name (*PyOpenColorIO.GpuShaderCreator.TextureDimensions property*), 367
name (*PyOpenColorIO.GpuShaderDesc.TextureDimensions property*), 377
NamedTransform (*class in PyOpenColorIO*), 343
NamedTransform.NamedTransformAliasIterator (*class in PyOpenColorIO*), 343
NAMEDTRANSFORM_ACTIVE (*PyOpenColorIO.NamedTransformVisibility attribute*), 269
NAMEDTRANSFORM_ALL (*PyOpenColorIO.NamedTransformVisibility attribute*), 270
NAMEDTRANSFORM_INACTIVE (*PyOpenColorIO.NamedTransformVisibility attribute*), 270
NamedTransformCategoryIterator (*class in PyOpenColorIO.NamedTransform*), 344
NamedTransformIterator (*class in PyOpenColorIO.Config*), 232

NamedTransformNameIterator (*class in PyOpenColorIO.Config*), 231
NamedTransformVisibility (*class in PyOpenColorIO*), 269
NEGATIVE_CLAMP (*PyOpenColorIO.NegativeStyle attribute*), 289
NEGATIVE_LINEAR (*PyOpenColorIO.NegativeStyle attribute*), 289
NEGATIVE_MIRROR (*PyOpenColorIO.NegativeStyle attribute*), 289
NEGATIVE_PASS_THRU (*PyOpenColorIO.NegativeStyle attribute*), 289
NegativeStyle (*class in PyOpenColorIO*), 289
NegativeStyleFromString() (*in module PyOpenColorIO*), 311
NegativeStyleToString() (*in module PyOpenColorIO*), 311
NoClampBlack (*PyOpenColorIO.GradingPrimary attribute*), 313
NoClampWhite (*PyOpenColorIO.GradingPrimary attribute*), 313

O

OCIO, 20, 94, 210
OCIO_INACTIVE_COLORSPACES, 57
offset (*PyOpenColorIO.GradingPrimary property*), 314
OpenColorIO_v2_5::Allocation (*C++ enum*), 280
OpenColorIO_v2_5::Allocation::ALLOCATION_LG2 (*C++ enumerator*), 280
OpenColorIO_v2_5::Allocation::ALLOCATION_UNIFORM (*C++ enumerator*), 280
OpenColorIO_v2_5::Allocation::ALLOCATION_UNKNOWN (*C++ enumerator*), 280
OpenColorIO_v2_5::AllocationFromString (*C++ function*), 308
OpenColorIO_v2_5::AllocationToString (*C++ function*), 308
OpenColorIO_v2_5::AllocationTransform (*C++ class*), 390
OpenColorIO_v2_5::AllocationTransform::~AllocationTransform (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::Create (*C++ function*), 391
OpenColorIO_v2_5::AllocationTransform::createEditableCopy (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::getAllocation (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::getDirection (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::getNumVars (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::getTransformType (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::getVars (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::operator= (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::setAllocation (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::setDirection (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::setVars (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransform::validate (*C++ function*), 390
OpenColorIO_v2_5::AllocationTransformRcPtr (*C++ type*), 391
OpenColorIO_v2_5::AutoStride (*C++ member*), 336
OpenColorIO_v2_5::Baker (*C++ class*), 196
OpenColorIO_v2_5::Baker::~Baker (*C++ function*), 198
OpenColorIO_v2_5::Baker::bake (*C++ function*), 197
OpenColorIO_v2_5::Baker::Baker (*C++ function*), 197
OpenColorIO_v2_5::Baker::Create (*C++ function*), 198
OpenColorIO_v2_5::Baker::createEditableCopy (*C++ function*), 196
OpenColorIO_v2_5::Baker::getConfig (*C++ function*), 196
OpenColorIO_v2_5::Baker::getCubeSize (*C++ function*), 197
OpenColorIO_v2_5::Baker::getDisplay (*C++ function*), 197
OpenColorIO_v2_5::Baker::getFormat (*C++ function*), 196
OpenColorIO_v2_5::Baker::getFormatExtensionByIndex (*C++ function*), 198
OpenColorIO_v2_5::Baker::getFormatMetadata (*C++ function*), 197
OpenColorIO_v2_5::Baker::getFormatNameByIndex (*C++ function*), 198
OpenColorIO_v2_5::Baker::getInputSpace (*C++ function*), 197
OpenColorIO_v2_5::Baker::getLooks (*C++ function*), 197
OpenColorIO_v2_5::Baker::getNumFormats (*C++ function*), 198
OpenColorIO_v2_5::Baker::getShaperSize (*C++ function*), 197
OpenColorIO_v2_5::Baker::getShaperSpace (*C++ function*), 197
OpenColorIO_v2_5::Baker::getTargetSpace (*C++ function*), 197
OpenColorIO_v2_5::Baker::getView (*C++ function*), 197

OpenColorIO_v2_5::Baker::operator= (C++ function), 198
OpenColorIO_v2_5::Baker::setConfig (C++ function), 196
OpenColorIO_v2_5::Baker::setCubeSize (C++ function), 197
OpenColorIO_v2_5::Baker::setDisplayView (C++ function), 197
OpenColorIO_v2_5::Baker::setFormat (C++ function), 196
OpenColorIO_v2_5::Baker::setInputSpace (C++ function), 197
OpenColorIO_v2_5::Baker::setLooks (C++ function), 197
OpenColorIO_v2_5::Baker::setShaperSize (C++ function), 197
OpenColorIO_v2_5::Baker::setShaperSpace (C++ function), 197
OpenColorIO_v2_5::Baker::setTargetSpace (C++ function), 197
OpenColorIO_v2_5::BakerRcPtr (C++ type), 198
OpenColorIO_v2_5::BitDepth (C++ enum), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_F16 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_F32 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT10 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT12 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT14 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT16 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT32 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UINT8 (C++ enumerator), 278
OpenColorIO_v2_5::BitDepth::BIT_DEPTH_UNKNOWN (C++ enumerator), 278
OpenColorIO_v2_5::BitDepthFromString (C++ function), 308
OpenColorIO_v2_5::BitDepthIsFloat (C++ function), 308
OpenColorIO_v2_5::BitDepthToInt (C++ function), 308
OpenColorIO_v2_5::BitDepthToString (C++ function), 308
OpenColorIO_v2_5::BoolFromString (C++ function), 307
OpenColorIO_v2_5::BoolToString (C++ function), 307
OpenColorIO_v2_5::BuiltinTransform (C++ class), 391
OpenColorIO_v2_5::BuiltinTransform::~BuiltinTransform (C++ function), 392
OpenColorIO_v2_5::BuiltinTransform::Create (C++ function), 392
OpenColorIO_v2_5::BuiltinTransform::getDescription (C++ function), 392
OpenColorIO_v2_5::BuiltinTransform::getStyle (C++ function), 392
OpenColorIO_v2_5::BuiltinTransform::getTransformType (C++ function), 392
OpenColorIO_v2_5::BuiltinTransform::setStyle (C++ function), 392
OpenColorIO_v2_5::BuiltinTransformRcPtr (C++ type), 392
OpenColorIO_v2_5::BuiltinTransformRegistry (C++ class), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::BuiltinTransfo (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::Get (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::getBuiltinDesc (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::getBuiltinStyl (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::getNumBuiltins (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistry::operator= (C++ function), 393
OpenColorIO_v2_5::BuiltinTransformRegistryRcPtr (C++ type), 393
OpenColorIO_v2_5::CDLStyle (C++ enum), 288
OpenColorIO_v2_5::CDLStyle::CDL_ASC (C++ enumerator), 289
OpenColorIO_v2_5::CDLStyle::CDL_NO_CLAMP (C++ enumerator), 289
OpenColorIO_v2_5::CDLStyle::CDL_TRANSFORM_DEFAULT (C++ enumerator), 289
OpenColorIO_v2_5::CDLStyleFromString (C++ function), 309
OpenColorIO_v2_5::CDLStyleToString (C++ function), 309
OpenColorIO_v2_5::CDLTransform (C++ class), 395
OpenColorIO_v2_5::CDLTransform::~CDLTransform (C++ function), 396
OpenColorIO_v2_5::CDLTransform::CDLTransform (C++ function), 396
OpenColorIO_v2_5::CDLTransform::Create (C++ function), 396
OpenColorIO_v2_5::CDLTransform::CreateFromFile (C++ function), 396
OpenColorIO_v2_5::CDLTransform::CreateGroupFromFile (C++ function), 396
OpenColorIO_v2_5::CDLTransform::equals (C++ function), 395

OpenColorIO_v2_5::CDLTransform::getFirstSOPDescription (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getFormatMetadata (C++ function), 395
OpenColorIO_v2_5::CDLTransform::getID (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getOffset (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getPower (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getSat (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getSatLumaCoefs (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getSlope (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getSOP (C++ function), 396
OpenColorIO_v2_5::CDLTransform::getStyle (C++ function), 395
OpenColorIO_v2_5::CDLTransform::getTransformType (C++ function), 395
OpenColorIO_v2_5::CDLTransform::operator= (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setFirstSOPDescription (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setID (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setOffset (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setPower (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setSat (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setSlope (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setSOP (C++ function), 396
OpenColorIO_v2_5::CDLTransform::setStyle (C++ function), 395
OpenColorIO_v2_5::CDLTransformRcPtr (C++ type), 397
OpenColorIO_v2_5::ChannelOrdering (C++ enum), 279
OpenColorIO_v2_5::ChannelOrdering::CHANNEL_ORDERING_ABGR (C++ enumerator), 280
OpenColorIO_v2_5::ChannelOrdering::CHANNEL_ORDERING_BGR (C++ enumerator), 280
OpenColorIO_v2_5::ChannelOrdering::CHANNEL_ORDERING_BGRA (C++ enumerator), 280
OpenColorIO_v2_5::ChannelOrdering::CHANNEL_ORDERING_RGB (C++ enumerator), 280
OpenColorIO_v2_5::ChannelOrdering::CHANNEL_ORDERING_RGBA (C++ enumerator), 280
OpenColorIO_v2_5::ClearAllCaches (C++ function), 303
OpenColorIO_v2_5::ColorSpace (C++ class), 202
OpenColorIO_v2_5::ColorSpace::~ColorSpace (C++ function), 205
OpenColorIO_v2_5::ColorSpace::addAlias (C++ function), 202
OpenColorIO_v2_5::ColorSpace::addCategory (C++ function), 203
OpenColorIO_v2_5::ColorSpace::clearAliases (C++ function), 202
OpenColorIO_v2_5::ColorSpace::clearCategories (C++ function), 204
OpenColorIO_v2_5::ColorSpace::ColorSpace (C++ function), 205
OpenColorIO_v2_5::ColorSpace::Create (C++ function), 205
OpenColorIO_v2_5::ColorSpace::createEditableCopy (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getAlias (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getAllocation (C++ function), 204
OpenColorIO_v2_5::ColorSpace::getAllocationNumVars (C++ function), 205
OpenColorIO_v2_5::ColorSpace::getAllocationVars (C++ function), 205
OpenColorIO_v2_5::ColorSpace::getBitDepth (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getCategory (C++ function), 204
OpenColorIO_v2_5::ColorSpace::getDescription (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getEncoding (C++ function), 204
OpenColorIO_v2_5::ColorSpace::getEqualityGroup (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getFamily (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getInterchangeAttribute (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getInterchangeAttributes (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getInteropID (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getName (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getNumAliases (C++ function), 202
OpenColorIO_v2_5::ColorSpace::getNumCategories (C++ function), 204
OpenColorIO_v2_5::ColorSpace::getReferenceSpaceType (C++ function), 203
OpenColorIO_v2_5::ColorSpace::getTransform

(C++ function), 205
OpenColorIO_v2_5::ColorSpace::hasAlias (C++ function), 202
OpenColorIO_v2_5::ColorSpace::hasCategory (C++ function), 203
OpenColorIO_v2_5::ColorSpace::isData (C++ function), 204
OpenColorIO_v2_5::ColorSpace::operator= (C++ function), 205
OpenColorIO_v2_5::ColorSpace::removeAlias (C++ function), 202
OpenColorIO_v2_5::ColorSpace::removeCategory (C++ function), 204
OpenColorIO_v2_5::ColorSpace::setAllocation (C++ function), 205
OpenColorIO_v2_5::ColorSpace::setAllocationVars (C++ function), 205
OpenColorIO_v2_5::ColorSpace::setBitDepth (C++ function), 203
OpenColorIO_v2_5::ColorSpace::setDescription (C++ function), 203
OpenColorIO_v2_5::ColorSpace::setEncoding (C++ function), 204
OpenColorIO_v2_5::ColorSpace::setEqualityGroup (C++ function), 202
OpenColorIO_v2_5::ColorSpace::setFamily (C++ function), 202
OpenColorIO_v2_5::ColorSpace::setInterchangeAttribute (C++ function), 203
OpenColorIO_v2_5::ColorSpace::setInteropID (C++ function), 203
OpenColorIO_v2_5::ColorSpace::setIsData (C++ function), 204
OpenColorIO_v2_5::ColorSpace::setName (C++ function), 202
OpenColorIO_v2_5::ColorSpace::setTransform (C++ function), 205
OpenColorIO_v2_5::ColorSpaceDirection (C++ enum), 271
OpenColorIO_v2_5::ColorSpaceDirection::COLORSPACE_DIR_FROM_REFERENCE (C++ enumerator), 271
OpenColorIO_v2_5::ColorSpaceDirection::COLORSPACE_DIR_TO_REFERENCE (C++ enumerator), 271
OpenColorIO_v2_5::ColorSpaceMenuHelper (C++ class), 184
OpenColorIO_v2_5::ColorSpaceMenuHelper::~ColorSpaceMenuHelper (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::ColorSpaceMenuHelper (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::Create (C++ function), 186
OpenColorIO_v2_5::ColorSpaceMenuHelper::getDescription (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getFamily (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getHierarchyLevel (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getIndexFromName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getIndexFromUIName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getNameFromUIName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getNumColorSpaces (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getNumHierarchyLev (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getUIName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::getUINameFromName (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelper::operator= (C++ function), 185
OpenColorIO_v2_5::ColorSpaceMenuHelperRcPtr (C++ type), 186
OpenColorIO_v2_5::ColorSpaceMenuParameters (C++ class), 181
OpenColorIO_v2_5::ColorSpaceMenuParameters::~ColorSpaceMen (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::addColorSpace (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::clearAddedColo (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::Create (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::getAddedColorS (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::getAppCategori (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getConfig (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getEncodings (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getIncludeColo (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getIncludeName (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getIncludeRole (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::getNumAddedCol (C++ function), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::getRole (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getSearchRefer (C++ function), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getTreatNoCate

(*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::getUseColorSpaces (*C++ function*), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::setAppCategories (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setConfig (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setEncoding (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setIncludeColorSpaces (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setIncludeNamedTransforms (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setIncludeRoles (*C++ function*), 184
OpenColorIO_v2_5::ColorSpaceMenuParameters::setRefCS (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setSearchReferenceSpaceType (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setUseAliases (*C++ function*), 183
OpenColorIO_v2_5::ColorSpaceMenuParameters::setUseColorSpaces (*C++ function*), 184
OpenColorIO_v2_5::ColorSpaceRcPtr (*C++ type*), 205
OpenColorIO_v2_5::ColorSpaceSet (*C++ class*), 208
OpenColorIO_v2_5::ColorSpaceSet::~ColorSpaceSet (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::addColorSpace (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::addColorSpaces (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::clearColorSpaces (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::Create (*C++ function*), 210
OpenColorIO_v2_5::ColorSpaceSet::createEditableCopy (*C++ function*), 208
OpenColorIO_v2_5::ColorSpaceSet::getColorSpace (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::getColorSpaceByIndex (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::getColorSpaceNames (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::getColorSpaceNameByIndex (*C++ function*), 208
OpenColorIO_v2_5::ColorSpaceSet::getNumColorSpaces (*C++ function*), 208
OpenColorIO_v2_5::ColorSpaceSet::hasColorSpace (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::operator!= (*C++ function*), 208
OpenColorIO_v2_5::ColorSpaceSet::operator== (*C++ function*), 208
OpenColorIO_v2_5::ColorSpaceSet::removeColorSpace (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSet::removeColorSpaces (*C++ function*), 209
OpenColorIO_v2_5::ColorSpaceSetRcPtr (*C++ type*), 210
OpenColorIO_v2_5::ColorSpaceTransform (*C++ class*), 397
OpenColorIO_v2_5::ColorSpaceTransform::~ColorSpaceTransform (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::Create (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::createEditableCopy (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::getDataBypass (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::getDirection (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::getDst (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::getSrc (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::getTransformType (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::operator= (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::setDataBypass (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::setDirection (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::setDst (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::setSrc (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransform::validate (*C++ function*), 398
OpenColorIO_v2_5::ColorSpaceTransformRcPtr (*C++ type*), 398
OpenColorIO_v2_5::ColorSpaceVisibility (*C++ enum*), 270
OpenColorIO_v2_5::ColorSpaceVisibility::COLORSPACE_ACTIVE (*C++ enumerator*), 270
OpenColorIO_v2_5::ColorSpaceVisibility::COLORSPACE_ALL (*C++ enumerator*), 270
OpenColorIO_v2_5::ColorSpaceVisibility::COLORSPACE_INACTIVE (*C++ enumerator*), 270
OpenColorIO_v2_5::CombineTransformDirections (*C++ function*), 308
OpenColorIO_v2_5::ComputeHashFunction (*C++ type*), 305
OpenColorIO_v2_5::Config (*C++ class*), 232
OpenColorIO_v2_5::Config::~Config (*C++ function*), 247

OpenColorIO_v2_5::Config::addActiveDisplay (C++ function), 242
OpenColorIO_v2_5::Config::addActiveView (C++ function), 243
OpenColorIO_v2_5::Config::addColorSpace (C++ function), 236
OpenColorIO_v2_5::Config::addDisplaySharedView (C++ function), 240
OpenColorIO_v2_5::Config::addDisplayView (C++ function), 239, 240
OpenColorIO_v2_5::Config::addEnvironmentVar (C++ function), 234
OpenColorIO_v2_5::Config::addLook (C++ function), 244
OpenColorIO_v2_5::Config::addNamedTransform (C++ function), 244
OpenColorIO_v2_5::Config::addSearchPath (C++ function), 235
OpenColorIO_v2_5::Config::addSharedView (C++ function), 238
OpenColorIO_v2_5::Config::addViewTransform (C++ function), 244
OpenColorIO_v2_5::Config::addVirtualDisplaySharedView (C++ function), 240
OpenColorIO_v2_5::Config::addVirtualDisplayView (C++ function), 240
OpenColorIO_v2_5::Config::archive (C++ function), 247
OpenColorIO_v2_5::Config::AreViewsEqual (C++ function), 250
OpenColorIO_v2_5::Config::AreVirtualViewsEqual (C++ function), 250
OpenColorIO_v2_5::Config::clearActiveDisplays (C++ function), 242
OpenColorIO_v2_5::Config::clearActiveViews (C++ function), 243
OpenColorIO_v2_5::Config::clearColorSpaces (C++ function), 237
OpenColorIO_v2_5::Config::clearDisplays (C++ function), 240
OpenColorIO_v2_5::Config::clearEnvironmentVars (C++ function), 234
OpenColorIO_v2_5::Config::clearLooks (C++ function), 244
OpenColorIO_v2_5::Config::clearNamedTransforms (C++ function), 245
OpenColorIO_v2_5::Config::clearProcessorCache (C++ function), 246
OpenColorIO_v2_5::Config::clearSearchPaths (C++ function), 235
OpenColorIO_v2_5::Config::clearSharedViews (C++ function), 239
OpenColorIO_v2_5::Config::clearViewTransforms (C++ function), 244
OpenColorIO_v2_5::Config::clearVirtualDisplay (C++ function), 241
OpenColorIO_v2_5::Config::Config (C++ function), 247
OpenColorIO_v2_5::Config::Create (C++ function), 248
OpenColorIO_v2_5::Config::createEditableCopy (C++ function), 233
OpenColorIO_v2_5::Config::CreateFromBuiltinConfig (C++ function), 249
OpenColorIO_v2_5::Config::CreateFromConfigIOProxy (C++ function), 248
OpenColorIO_v2_5::Config::CreateFromEnv (C++ function), 248
OpenColorIO_v2_5::Config::CreateFromFile (C++ function), 248
OpenColorIO_v2_5::Config::CreateFromStream (C++ function), 248
OpenColorIO_v2_5::Config::CreateRaw (C++ function), 248
OpenColorIO_v2_5::Config::filepathOnlyMatchesDefaultRule (C++ function), 245
OpenColorIO_v2_5::Config::getActiveDisplay (C++ function), 242
OpenColorIO_v2_5::Config::getActiveDisplays (C++ function), 242
OpenColorIO_v2_5::Config::getActiveView (C++ function), 242
OpenColorIO_v2_5::Config::getActiveViews (C++ function), 242
OpenColorIO_v2_5::Config::getCacheID (C++ function), 234
OpenColorIO_v2_5::Config::getCanonicalName (C++ function), 236
OpenColorIO_v2_5::Config::getColorSpace (C++ function), 236
OpenColorIO_v2_5::Config::getColorSpaceFromFilepath (C++ function), 245
OpenColorIO_v2_5::Config::getColorSpaceNameByIndex (C++ function), 235, 236
OpenColorIO_v2_5::Config::getColorSpaces (C++ function), 235
OpenColorIO_v2_5::Config::getConfigIOProxy (C++ function), 247
OpenColorIO_v2_5::Config::getCurrentContext (C++ function), 234
OpenColorIO_v2_5::Config::getDefaultDisplay (C++ function), 239
OpenColorIO_v2_5::Config::GetDefaultFamilySeparator (C++ function), 249
OpenColorIO_v2_5::Config::getDefaultLumaCoefs (C++ function), 243
OpenColorIO_v2_5::Config::getDefaultSceneToDisplayViewTrar (C++ function), 244

OpenColorIO_v2_5::Config::getDefaultView (C++ function), 239
OpenColorIO_v2_5::Config::getDefaultViewTransformName (C++ function), 244
OpenColorIO_v2_5::Config::getDescription (C++ function), 234
OpenColorIO_v2_5::Config::getDisplay (C++ function), 239
OpenColorIO_v2_5::Config::getDisplayAll (C++ function), 243
OpenColorIO_v2_5::Config::getDisplayAllByName (C++ function), 243
OpenColorIO_v2_5::Config::getDisplayViewColorSpaceName (C++ function), 239
OpenColorIO_v2_5::Config::getDisplayViewDescription (C++ function), 239
OpenColorIO_v2_5::Config::getDisplayViewLooks (C++ function), 239
OpenColorIO_v2_5::Config::getDisplayViewRule (C++ function), 239
OpenColorIO_v2_5::Config::getDisplayViewTransformName (C++ function), 239
OpenColorIO_v2_5::Config::getEnvironmentMode (C++ function), 234
OpenColorIO_v2_5::Config::getEnvironmentVarDefault (C++ function), 234
OpenColorIO_v2_5::Config::getEnvironmentVarNameByIndex (C++ function), 234
OpenColorIO_v2_5::Config::getFamilySeparator (C++ function), 233
OpenColorIO_v2_5::Config::getFileRules (C++ function), 245
OpenColorIO_v2_5::Config::getInactiveColorSpaces (C++ function), 237
OpenColorIO_v2_5::Config::getIndexForColorSpace (C++ function), 236
OpenColorIO_v2_5::Config::getIndexForNamedTransform (C++ function), 244
OpenColorIO_v2_5::Config::getLook (C++ function), 243
OpenColorIO_v2_5::Config::getLookNameByIndex (C++ function), 244
OpenColorIO_v2_5::Config::getMajorVersion (C++ function), 233
OpenColorIO_v2_5::Config::getMinorVersion (C++ function), 233
OpenColorIO_v2_5::Config::getName (C++ function), 233
OpenColorIO_v2_5::Config::getNamedTransform (C++ function), 244
OpenColorIO_v2_5::Config::getNamedTransformNameByIndex (C++ function), 244
OpenColorIO_v2_5::Config::getNumActiveDisplays (C++ function), 242
OpenColorIO_v2_5::Config::getNumActiveViews (C++ function), 242
OpenColorIO_v2_5::Config::getNumColorSpaces (C++ function), 235
OpenColorIO_v2_5::Config::getNumDisplays (C++ function), 239
OpenColorIO_v2_5::Config::getNumDisplaysAll (C++ function), 243
OpenColorIO_v2_5::Config::getNumEnvironmentVars (C++ function), 234
OpenColorIO_v2_5::Config::getNumLooks (C++ function), 244
OpenColorIO_v2_5::Config::getNumNamedTransforms (C++ function), 244
OpenColorIO_v2_5::Config::getNumRoles (C++ function), 238
OpenColorIO_v2_5::Config::getNumSearchPaths (C++ function), 235
OpenColorIO_v2_5::Config::getNumViews (C++ function), 239, 243
OpenColorIO_v2_5::Config::getNumViewTransforms (C++ function), 244
OpenColorIO_v2_5::Config::getProcessor (C++ function), 245, 246
OpenColorIO_v2_5::Config::getProcessorCacheFlags (C++ function), 246
OpenColorIO_v2_5::Config::GetProcessorFromBuiltinColorSpac (C++ function), 251
OpenColorIO_v2_5::Config::GetProcessorFromConfigs (C++ function), 251, 252
OpenColorIO_v2_5::Config::GetProcessorToBuiltinColorSpace (C++ function), 251
OpenColorIO_v2_5::Config::getRoleColorSpace (C++ function), 238
OpenColorIO_v2_5::Config::getRoleName (C++ function), 238
OpenColorIO_v2_5::Config::getSearchPath (C++ function), 235
OpenColorIO_v2_5::Config::getView (C++ function), 239, 243
OpenColorIO_v2_5::Config::getViewingRules (C++ function), 243
OpenColorIO_v2_5::Config::getViewTransform (C++ function), 244
OpenColorIO_v2_5::Config::getViewTransformNameByIndex (C++ function), 244
OpenColorIO_v2_5::Config::getVirtualDisplayNumViews (C++ function), 241
OpenColorIO_v2_5::Config::getVirtualDisplayView (C++ function), 241
OpenColorIO_v2_5::Config::getVirtualDisplayViewColorSpaceN (C++ function), 241
OpenColorIO_v2_5::Config::getVirtualDisplayViewDescriptior (C++ function), 241

OpenColorIO_v2_5::Config::getVirtualDisplayViewCount (C++ function), 241
OpenColorIO_v2_5::Config::getVirtualDisplayViewName (C++ function), 241
OpenColorIO_v2_5::Config::getVirtualDisplayViewTransformName (C++ function), 241
OpenColorIO_v2_5::Config::getWorkingDir (C++ function), 235
OpenColorIO_v2_5::Config::hasRole (C++ function), 238
OpenColorIO_v2_5::Config::hasView (C++ function), 239
OpenColorIO_v2_5::Config::hasVirtualView (C++ function), 240
OpenColorIO_v2_5::Config::IdentifyBuiltinColorSpace (C++ function), 249
OpenColorIO_v2_5::Config::IdentifyInterchangeSpace (C++ function), 250
OpenColorIO_v2_5::Config::instantiateDisplayFromConfig (C++ function), 241
OpenColorIO_v2_5::Config::instantiateDisplayFromMemory (C++ function), 241
OpenColorIO_v2_5::Config::isArchivable (C++ function), 247
OpenColorIO_v2_5::Config::isColorSpaceLinear (C++ function), 237
OpenColorIO_v2_5::Config::isColorSpaceUsed (C++ function), 236
OpenColorIO_v2_5::Config::isDisplayTemporary (C++ function), 243
OpenColorIO_v2_5::Config::isInactiveColorSpace (C++ function), 237
OpenColorIO_v2_5::Config::isStrictParsingEnabled (C++ function), 245
OpenColorIO_v2_5::Config::isViewShared (C++ function), 238
OpenColorIO_v2_5::Config::isVirtualViewShared (C++ function), 240
OpenColorIO_v2_5::Config::loadEnvironment (C++ function), 235
OpenColorIO_v2_5::Config::operator= (C++ function), 247
OpenColorIO_v2_5::Config::parseColorSpaceFromString (C++ function), 245
OpenColorIO_v2_5::Config::removeActiveDisplay (C++ function), 242
OpenColorIO_v2_5::Config::removeActiveView (C++ function), 243
OpenColorIO_v2_5::Config::removeColorSpace (C++ function), 236
OpenColorIO_v2_5::Config::removeDisplayView (C++ function), 240
OpenColorIO_v2_5::Config::removeNamedTransform (C++ function), 245
OpenColorIO_v2_5::Config::removeSharedView (C++ function), 239
OpenColorIO_v2_5::Config::removeVirtualDisplayView (C++ function), 241
OpenColorIO_v2_5::Config::serialize (C++ function), 234
OpenColorIO_v2_5::Config::setActiveDisplays (C++ function), 241
OpenColorIO_v2_5::Config::setActiveViews (C++ function), 242
OpenColorIO_v2_5::Config::setConfigIOProxy (C++ function), 246
OpenColorIO_v2_5::Config::setDefaultLumaCoefs (C++ function), 243
OpenColorIO_v2_5::Config::setDefaultViewTransformName (C++ function), 244
OpenColorIO_v2_5::Config::setDescription (C++ function), 234
OpenColorIO_v2_5::Config::setDisplayTemporary (C++ function), 243
OpenColorIO_v2_5::Config::setEnvironmentMode (C++ function), 234
OpenColorIO_v2_5::Config::setFamilySeparator (C++ function), 233
OpenColorIO_v2_5::Config::setFileRules (C++ function), 245
OpenColorIO_v2_5::Config::setInactiveColorSpaces (C++ function), 237
OpenColorIO_v2_5::Config::setMajorVersion (C++ function), 233
OpenColorIO_v2_5::Config::setMinorVersion (C++ function), 233
OpenColorIO_v2_5::Config::setName (C++ function), 233
OpenColorIO_v2_5::Config::setProcessorCacheFlags (C++ function), 246
OpenColorIO_v2_5::Config::setRole (C++ function), 238
OpenColorIO_v2_5::Config::setSearchPath (C++ function), 235
OpenColorIO_v2_5::Config::setStrictParsingEnabled (C++ function), 245
OpenColorIO_v2_5::Config::setVersion (C++ function), 233
OpenColorIO_v2_5::Config::setViewingRules (C++ function), 243
OpenColorIO_v2_5::Config::setWorkingDir (C++ function), 235
OpenColorIO_v2_5::Config::upgradeToLatestVersion (C++ function), 233
OpenColorIO_v2_5::Config::validate (C++ function), 233
OpenColorIO_v2_5::ConfigRcPtr (C++ type), 253
OpenColorIO_v2_5::ConstAllocationTransformRcPtr

(*C++ type*), 391
OpenColorIO_v2_5::ConstBakerRcPtr (*C++ type*), 198
OpenColorIO_v2_5::ConstBuiltinTransformRcPtr (*C++ type*), 392
OpenColorIO_v2_5::ConstBuiltinTransformRegistryPtr (*C++ type*), 393
OpenColorIO_v2_5::ConstCDLTransformRcPtr (*C++ type*), 397
OpenColorIO_v2_5::ConstColorSpaceMenuHelperRcPtr (*C++ type*), 186
OpenColorIO_v2_5::ConstColorSpaceRcPtr (*C++ type*), 205
OpenColorIO_v2_5::ConstColorSpaceSetRcPtr (*C++ type*), 210
OpenColorIO_v2_5::ConstColorSpaceTransformRcPtr (*C++ type*), 398
OpenColorIO_v2_5::ConstConfigRcPtr (*C++ type*), 253
OpenColorIO_v2_5::ConstContextRcPtr (*C++ type*), 261
OpenColorIO_v2_5::ConstCPUProcessorRcPtr (*C++ type*), 355
OpenColorIO_v2_5::ConstDisplayViewTransformRcPtr (*C++ type*), 400
OpenColorIO_v2_5::ConstDynamicPropertyDoubleRcPtr (*C++ type*), 265
OpenColorIO_v2_5::ConstDynamicPropertyGradingPrimaryRcPtr (*C++ type*), 266
OpenColorIO_v2_5::ConstDynamicPropertyGradingRGBCurveRcPtr (*C++ type*), 266
OpenColorIO_v2_5::ConstDynamicPropertyGradingToneRcPtr (*C++ type*), 267
OpenColorIO_v2_5::ConstDynamicPropertyRcPtr (*C++ type*), 264
OpenColorIO_v2_5::ConstExponentTransformRcPtr (*C++ type*), 402
OpenColorIO_v2_5::ConstExponentWithLinearTransformRcPtr (*C++ type*), 404
OpenColorIO_v2_5::ConstExposureContrastTransformRcPtr (*C++ type*), 408
OpenColorIO_v2_5::ConstFileRulesRcPtr (*C++ type*), 362
OpenColorIO_v2_5::ConstFileTransformRcPtr (*C++ type*), 411
OpenColorIO_v2_5::ConstFixedFunctionTransformRcPtr (*C++ type*), 412
OpenColorIO_v2_5::ConstGPUProcessorRcPtr (*C++ type*), 355
OpenColorIO_v2_5::ConstGpuShaderCreatorRcPtr (*C++ type*), 374
OpenColorIO_v2_5::ConstGpuShaderDescRcPtr (*C++ type*), 387
OpenColorIO_v2_5::ConstGradingBSplineCurveRcPtr (*C++ type*), 329
OpenColorIO_v2_5::ConstGradingHueCurveRcPtr (*C++ type*), 326
OpenColorIO_v2_5::ConstGradingHueCurveTransformRcPtr (*C++ type*), 324
OpenColorIO_v2_5::ConstGradingPrimaryTransformRcPtr (*C++ type*), 313
OpenColorIO_v2_5::ConstGradingRGBCurveRcPtr (*C++ type*), 320
OpenColorIO_v2_5::ConstGradingRGBCurveTransformRcPtr (*C++ type*), 319
OpenColorIO_v2_5::ConstGradingToneTransformRcPtr (*C++ type*), 332
OpenColorIO_v2_5::ConstGroupTransformRcPtr (*C++ type*), 415
OpenColorIO_v2_5::ConstImageDescRcPtr (*C++ type*), 336
OpenColorIO_v2_5::ConstLegacyViewingPipelineRcPtr (*C++ type*), 190
OpenColorIO_v2_5::ConstLogAffineTransformRcPtr (*C++ type*), 417
OpenColorIO_v2_5::ConstLogCameraTransformRcPtr (*C++ type*), 420
OpenColorIO_v2_5::ConstLogTransformRcPtr (*C++ type*), 422
OpenColorIO_v2_5::ConstLookRcPtr (*C++ type*), 343
OpenColorIO_v2_5::ConstLookTransformRcPtr (*C++ type*), 424
OpenColorIO_v2_5::ConstLut1DTransformRcPtr (*C++ type*), 427
OpenColorIO_v2_5::ConstLut3DTransformRcPtr (*C++ type*), 429
OpenColorIO_v2_5::ConstMatrixTransformRcPtr (*C++ type*), 433
OpenColorIO_v2_5::ConstMixingColorSpaceManagerRcPtr (*C++ type*), 194
OpenColorIO_v2_5::ConstNamedTransformRcPtr (*C++ type*), 346
OpenColorIO_v2_5::ConstProcessorMetadataRcPtr (*C++ type*), 357
OpenColorIO_v2_5::ConstProcessorRcPtr (*C++ type*), 351
OpenColorIO_v2_5::ConstRangeTransformRcPtr (*C++ type*), 437
OpenColorIO_v2_5::ConstSystemMonitorsRcPtr (*C++ type*), 388
OpenColorIO_v2_5::ConstTransformRcPtr (*C++ type*), 389
OpenColorIO_v2_5::ConstViewingRulesRcPtr (*C++ type*), 366
OpenColorIO_v2_5::ConstViewTransformRcPtr (*C++ type*), 440
OpenColorIO_v2_5::Context (*C++ class*), 259

OpenColorIO_v2_5::Context::~Context (*C++ function*), 261
OpenColorIO_v2_5::Context::addSearchPath (*C++ function*), 260
OpenColorIO_v2_5::Context::addStringVars (*C++ function*), 260
OpenColorIO_v2_5::Context::clearSearchPaths (*C++ function*), 260
OpenColorIO_v2_5::Context::clearStringVars (*C++ function*), 260
OpenColorIO_v2_5::Context::Context (*C++ function*), 261
OpenColorIO_v2_5::Context::Create (*C++ function*), 261
OpenColorIO_v2_5::Context::createEditableCopy (*C++ function*), 260
OpenColorIO_v2_5::Context::getCacheID (*C++ function*), 260
OpenColorIO_v2_5::Context::getConfigIOProxy (*C++ function*), 261
OpenColorIO_v2_5::Context::getEnvironmentMode (*C++ function*), 260
OpenColorIO_v2_5::Context::getNumSearchPaths (*C++ function*), 260
OpenColorIO_v2_5::Context::getNumStringVars (*C++ function*), 260
OpenColorIO_v2_5::Context::getSearchPath (*C++ function*), 260
OpenColorIO_v2_5::Context::getStringVar (*C++ function*), 260
OpenColorIO_v2_5::Context::getStringVarByIndex (*C++ function*), 260
OpenColorIO_v2_5::Context::getStringVarNameByIndex (*C++ function*), 260
OpenColorIO_v2_5::Context::getWorkingDir (*C++ function*), 260
OpenColorIO_v2_5::Context::loadEnvironment (*C++ function*), 260
OpenColorIO_v2_5::Context::operator= (*C++ function*), 261
OpenColorIO_v2_5::Context::resolveFileLocation (*C++ function*), 260, 261
OpenColorIO_v2_5::Context::resolveStringVar (*C++ function*), 260
OpenColorIO_v2_5::Context::setConfigIOProxy (*C++ function*), 261
OpenColorIO_v2_5::Context::setEnvironmentMode (*C++ function*), 260
OpenColorIO_v2_5::Context::setSearchPath (*C++ function*), 260
OpenColorIO_v2_5::Context::setStringVar (*C++ function*), 260
OpenColorIO_v2_5::Context::setWorkingDir (*C++ function*), 260
OpenColorIO_v2_5::ContextRcPtr (*C++ type*), 261
OpenColorIO_v2_5::CPUProcessor (*C++ class*), 353
OpenColorIO_v2_5::CPUProcessor::~CPUProcessor (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::apply (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::applyRGB (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::applyRGBA (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::CPUProcessor (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::getCacheID (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::getDynamicProperty (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::getInputBitDepth (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::getOutputBitDepth (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::hasChannelCrosstalk (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::hasDynamicProperty (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::isDynamic (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::isIdentity (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::isNoOp (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessor::operator= (*C++ function*), 354
OpenColorIO_v2_5::CPUProcessorRcPtr (*C++ type*), 355
OpenColorIO_v2_5::DisplayViewHelpers (*C++ type*), 186
OpenColorIO_v2_5::DisplayViewHelpers::AddDisplayView (*C++ function*), 187
OpenColorIO_v2_5::DisplayViewHelpers::GetIdentityProcessor (*C++ function*), 187
OpenColorIO_v2_5::DisplayViewHelpers::GetProcessor (*C++ function*), 187
OpenColorIO_v2_5::DisplayViewHelpers::RemoveDisplayView (*C++ function*), 187
OpenColorIO_v2_5::DisplayViewTransform (*C++ class*), 399
OpenColorIO_v2_5::DisplayViewTransform::~DisplayViewTransf (*C++ function*), 400
OpenColorIO_v2_5::DisplayViewTransform::Create (*C++ function*), 400
OpenColorIO_v2_5::DisplayViewTransform::createEditableCopy (*C++ function*), 399
OpenColorIO_v2_5::DisplayViewTransform::getDataBypass (*C++ function*), 400

OpenColorIO_v2_5::DisplayViewTransform::getDirection (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::getDisplay (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::getLooksBypass (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::getSrc (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::getTransformType (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::getView (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::setDataBypass (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::setDirection (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::setDisplay (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::setLooksBypass (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::setSrc (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransform::setView (C++ function), 400
OpenColorIO_v2_5::DisplayViewTransform::validate (C++ function), 399
OpenColorIO_v2_5::DisplayViewTransformRcPtr (C++ type), 400
OpenColorIO_v2_5::DynamicProperty (C++ class), 263
OpenColorIO_v2_5::DynamicProperty::~DynamicProperty (C++ function), 264
OpenColorIO_v2_5::DynamicProperty::DynamicProperty (C++ function), 264
OpenColorIO_v2_5::DynamicProperty::getType (C++ function), 264
OpenColorIO_v2_5::DynamicProperty::operator= (C++ function), 264
OpenColorIO_v2_5::DynamicPropertyDouble (C++ class), 265
OpenColorIO_v2_5::DynamicPropertyDouble::~DynamicPropertyDouble (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyDouble::DynamicPropertyDouble (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyDouble::getValue (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyDouble::operator= (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyDouble::setValue (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyDoubleRcPtr (C++ type), 265
OpenColorIO_v2_5::DynamicPropertyGradingPrimary (C++ class), 265
OpenColorIO_v2_5::DynamicPropertyGradingPrimary::~DynamicP (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingPrimary::DynamicPr (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingPrimary::getValue (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingPrimary::operator= (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingPrimary::setValue (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingPrimaryRcPtr (C++ type), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve (C++ class), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve::~Dynamic (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve::DynamicP (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve::getValue (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve::operator (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurve::setValue (C++ function), 266
OpenColorIO_v2_5::DynamicPropertyGradingRGBCurveRcPtr (C++ type), 266
OpenColorIO_v2_5::DynamicPropertyGradingTone (C++ class), 267
OpenColorIO_v2_5::DynamicPropertyGradingTone::~DynamicProp (C++ function), 267
OpenColorIO_v2_5::DynamicPropertyGradingTone::DynamicPrope (C++ function), 267
OpenColorIO_v2_5::DynamicPropertyGradingTone::getValue (C++ function), 267
OpenColorIO_v2_5::DynamicPropertyGradingTone::operator= (C++ function), 267
OpenColorIO_v2_5::DynamicPropertyGradingTone::setValue (C++ function), 267
OpenColorIO_v2_5::DynamicPropertyGradingToneRcPtr (C++ type), 267
OpenColorIO_v2_5::DynamicPropertyRcPtr (C++ type), 264
OpenColorIO_v2_5::DynamicPropertyType (C++ enum), 291
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_CO (C++ enumerator), 291
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_EX (C++ enumerator), 291
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_GA (C++ enumerator), 291
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_GR (C++ enumerator), 292
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_GR (C++ enumerator), 291

OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_GRADING_RGBCURVE (C++ enumerator), 292
OpenColorIO_v2_5::DynamicPropertyType::DYNAMIC_PROPERTY_GRADING_TONE (C++ enumerator), 292
OpenColorIO_v2_5::DynamicPropertyValue (C++ type), 264
OpenColorIO_v2_5::DynamicPropertyValue::AsDouble (C++ function), 264
OpenColorIO_v2_5::DynamicPropertyValue::AsGradingHueCurve (C++ function), 265
OpenColorIO_v2_5::DynamicPropertyValue::AsGradingPrimary (C++ function), 264
OpenColorIO_v2_5::DynamicPropertyValue::AsGradingRGBCurve (C++ function), 264
OpenColorIO_v2_5::DynamicPropertyValue::AsGradingTone (C++ function), 265
OpenColorIO_v2_5::DynamicPtrCast (C++ function), 307
OpenColorIO_v2_5::EnvironmentMode (C++ enum), 283
OpenColorIO_v2_5::EnvironmentMode::ENV_ENVIRONMENT_LOAD_ALL (C++ enumerator), 283
OpenColorIO_v2_5::EnvironmentMode::ENV_ENVIRONMENT_LOAD_PREDEFINED (C++ enumerator), 283
OpenColorIO_v2_5::EnvironmentMode::ENV_ENVIRONMENT_UNKNOWN (C++ enumerator), 283
OpenColorIO_v2_5::EnvironmentModeFromString (C++ function), 309
OpenColorIO_v2_5::EnvironmentModeToString (C++ function), 309
OpenColorIO_v2_5::Exception (C++ class), 299
OpenColorIO_v2_5::Exception::~Exception (C++ function), 299
OpenColorIO_v2_5::Exception::Exception (C++ function), 299
OpenColorIO_v2_5::Exception::operator= (C++ function), 299
OpenColorIO_v2_5::ExceptionMissingFile (C++ class), 299
OpenColorIO_v2_5::ExceptionMissingFile::~ExceptionMissingFile (C++ function), 300
OpenColorIO_v2_5::ExceptionMissingFile::ExceptionMissingFile (C++ function), 300
OpenColorIO_v2_5::ExceptionMissingFile::operator= (C++ function), 300
OpenColorIO_v2_5::ExponentTransform (C++ class), 401
OpenColorIO_v2_5::ExponentTransform::~ExponentTransform (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::Create (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::equals (C++ function), 401
OpenColorIO_v2_5::ExponentTransform::ExponentTransform (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::getFormatMetadata (C++ function), 401
OpenColorIO_v2_5::ExponentTransform::getNegativeStyle (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::getTransformType (C++ function), 401
OpenColorIO_v2_5::ExponentTransform::getValue (C++ function), 401
OpenColorIO_v2_5::ExponentTransform::operator= (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::setNegativeStyle (C++ function), 402
OpenColorIO_v2_5::ExponentTransform::setValue (C++ function), 402
OpenColorIO_v2_5::ExponentTransformRcPtr (C++ type), 402
OpenColorIO_v2_5::ExponentWithLinearTransform (C++ class), 403
OpenColorIO_v2_5::ExponentWithLinearTransform::~ExponentWithLinearTransform (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::Create (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::equals (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::ExponentWithLinearTransform (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::getFormatMetadata (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::getGamma (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::getNegativeStyle (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::getOffset (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::getTransformType (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::operator= (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::setGamma (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::setNegativeStyle (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransform::setOffset (C++ function), 404
OpenColorIO_v2_5::ExponentWithLinearTransformRcPtr (C++ type), 404
OpenColorIO_v2_5::ExposureContrastStyle (C++ enum), 288
OpenColorIO_v2_5::ExposureContrastStyle::EXPOSURE_CONTRAST (C++ enumerator), 288
OpenColorIO_v2_5::ExposureContrastStyle::EXPOSURE_CONTRAST (C++ enumerator), 288
OpenColorIO_v2_5::ExposureContrastStyle::EXPOSURE_CONTRAST

(C++ *enumerator*), 288
OpenColorIO_v2_5::ExposureContrastStyleFromString (C++ *function*), 311
OpenColorIO_v2_5::ExposureContrastStyleToString (C++ *function*), 310
OpenColorIO_v2_5::ExposureContrastTransform (C++ *class*), 406
OpenColorIO_v2_5::ExposureContrastTransform::~ExposureContrastTransform (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::Create (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::equals (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getContrast (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getExposure (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getFormatMetadata (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getGamma (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getLogExposureStep (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::getLogMidGray (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::getPivot (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getStyle (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::getTransformType (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::isContrastDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::isExposureDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::isGammaDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeContrastDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeContrastNonDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeExposureDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeExposureNonDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeGammaDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::makeGammaNonDynamic (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::setContrast (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::setExposure (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::setGamma (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransform::setLogExposureStep (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::setLogMidGray (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::setPivot (C++ *function*), 408
OpenColorIO_v2_5::ExposureContrastTransform::setStyle (C++ *function*), 407
OpenColorIO_v2_5::ExposureContrastTransformRcPtr (C++ *type*), 408
OpenColorIO_v2_5::FileRules (C++ *class*), 359
OpenColorIO_v2_5::FileRules::~FileRules (C++ *function*), 362
OpenColorIO_v2_5::FileRules::Create (C++ *function*), 362
OpenColorIO_v2_5::FileRules::createEditableCopy (C++ *function*), 360
OpenColorIO_v2_5::FileRules::decreaseRulePriority (C++ *function*), 361
OpenColorIO_v2_5::FileRules::DefaultRuleName (C++ *member*), 362
OpenColorIO_v2_5::FileRules::FilePathSearchRuleName (C++ *member*), 362
OpenColorIO_v2_5::FileRules::FileRules (C++ *function*), 362
OpenColorIO_v2_5::FileRules::getColorSpace (C++ *function*), 361
OpenColorIO_v2_5::FileRules::getCustomKeyName (C++ *function*), 361
OpenColorIO_v2_5::FileRules::getCustomKeyValue (C++ *function*), 361
OpenColorIO_v2_5::FileRules::getExtension (C++ *function*), 360
OpenColorIO_v2_5::FileRules::getIndexForRule (C++ *function*), 360
OpenColorIO_v2_5::FileRules::getName (C++ *function*), 360
OpenColorIO_v2_5::FileRules::getNumCustomKeys (C++ *function*), 361
OpenColorIO_v2_5::FileRules::getNumEntries (C++ *function*), 360
OpenColorIO_v2_5::FileRules::getPattern (C++ *function*), 360
OpenColorIO_v2_5::FileRules::getRegex (C++ *function*), 360
OpenColorIO_v2_5::FileRules::increaseRulePriority (C++ *function*), 361
OpenColorIO_v2_5::FileRules::insertPathSearchRule (C++ *function*), 361
OpenColorIO_v2_5::FileRules::insertRule (C++ *function*), 361
OpenColorIO_v2_5::FileRules::isDefault (C++ *function*), 361

OpenColorIO_v2_5::FileRules::operator= (*C++ function*), 362
OpenColorIO_v2_5::FileRules::removeRule (*C++ function*), 361
OpenColorIO_v2_5::FileRules::setColorSpace (*C++ function*), 361
OpenColorIO_v2_5::FileRules::setCustomKey (*C++ function*), 361
OpenColorIO_v2_5::FileRules::setDefaultRuleColorSpace (*C++ function*), 361
OpenColorIO_v2_5::FileRules::setExtension (*C++ function*), 360
OpenColorIO_v2_5::FileRules::setPattern (*C++ function*), 360
OpenColorIO_v2_5::FileRules::setRegex (*C++ function*), 361
OpenColorIO_v2_5::FileRulesRcPtr (*C++ type*), 362
OpenColorIO_v2_5::FileTransform (*C++ class*), 409
OpenColorIO_v2_5::FileTransform::~FileTransform (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::Create (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::createEditableCopy (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getCCCId (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getCDLStyle (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getDirection (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::GetFormatExtensionByIndex (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::GetFormatNameByIndex (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getInterpolation (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::GetNumFormats (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getSrc (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::getTransformType (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::IsFormatExtensionSupported (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::operator= (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::setCCCId (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::setCDLStyle (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::setDirection (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::setInterpolation (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::setSrc (*C++ function*), 410
OpenColorIO_v2_5::FileTransform::validate (*C++ function*), 410
OpenColorIO_v2_5::FileTransformRcPtr (*C++ type*), 411
OpenColorIO_v2_5::FixedFunctionStyle (*C++ enum*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_ACES_ (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_LIN_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_LIN_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_LIN_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_REC21 (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_RGB_T (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_RGB_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_RGB_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_RGB_T (*C++ enumerator*), 287
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_XYZ_T (*C++ enumerator*), 286
OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_XYZ_T (*C++ enumerator*), 286

OpenColorIO_v2_5::FixedFunctionStyle::FIXED_FUNCTION_LOGTOE_2_xy_XYZ_to_xyY (C++ enumerator), 286
OpenColorIO_v2_5::FixedFunctionStyleFromString (C++ function), 310
OpenColorIO_v2_5::FixedFunctionStyleToString (C++ function), 310
OpenColorIO_v2_5::FixedFunctionTransform (C++ class), 411
OpenColorIO_v2_5::FixedFunctionTransform::~FixedFunctionTransform (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::Create (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::equals (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::FixedFunctionTransform (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::getFormatMetadata (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::getNumParams (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::getParams (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::getStyle (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::getTransformType (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::operator= (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::setParams (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransform::setStyle (C++ function), 412
OpenColorIO_v2_5::FixedFunctionTransformRcPtr (C++ type), 412
OpenColorIO_v2_5::FormatMetadata (C++ class), 302
OpenColorIO_v2_5::FormatMetadata::~FormatMetadata (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::addAttribute (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::addChildElement (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::clear (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::FormatMetadata (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::getAttributeName (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getAttributeValue (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getChildElement (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getElementName (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getElementValue (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getID (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::getName (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::getNumAttributes (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::getNumChildrenElements (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::operator= (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::setElementName (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::setElementValue (C++ function), 302
OpenColorIO_v2_5::FormatMetadata::setID (C++ function), 303
OpenColorIO_v2_5::FormatMetadata::setName (C++ function), 303
OpenColorIO_v2_5::GetCurrentConfig (C++ function), 232
OpenColorIO_v2_5::GetEnvVariable (C++ function), 306
OpenColorIO_v2_5::GetInverseTransformDirection (C++ function), 308
OpenColorIO_v2_5::GetLoggingLevel (C++ function), 304
OpenColorIO_v2_5::GetVersion (C++ function), 304
OpenColorIO_v2_5::GetVersionHex (C++ function), 304
OpenColorIO_v2_5::GpuLanguage (C++ enum), 281
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_CG (C++ enumerator), 281
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_1_2 (C++ enumerator), 281
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_1_3 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_4_0 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_ES_1_0 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_ES_3_0 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_GLSL_VK_4_6 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_HLSL_DX11 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_HLSL_SM_5_0 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::GPU_LANGUAGE_MSL_2_0 (C++ enumerator), 282
OpenColorIO_v2_5::GpuLanguage::LANGUAGE_OSL_1 (C++ enumerator), 282

OpenColorIO_v2_5::GpuLanguageFromString (C++ function), 309
OpenColorIO_v2_5::GpuLanguageToString (C++ function), 309
OpenColorIO_v2_5::GPUProcessor (C++ class), 355
OpenColorIO_v2_5::GPUProcessor::~GPUProcessor (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::extractGpuShaderInfo (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::getCacheID (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::GPUProcessor (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::hasChannelCrosstalk (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::isNoOp (C++ function), 355
OpenColorIO_v2_5::GPUProcessor::operator= (C++ function), 355
OpenColorIO_v2_5::GPUProcessorRcPtr (C++ type), 355
OpenColorIO_v2_5::GpuShaderCreator (C++ class), 370
OpenColorIO_v2_5::GpuShaderCreator::~GpuShaderCreator (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::add3DTexture (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addDynamicProperty (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::addTexture (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::addToFunctionHeaderShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addToFunctionParameterShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addToFunctionShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addToHelperShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addToParameterShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addToTexelDeclareShaderCode (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::addUniform (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::begin (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::BoolGetter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreator::clone (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::createShaderText (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::DoubleGetter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreator::end (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::finalize (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::Float3Getter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreator::getAllowTexture1D (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::getCacheID (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getDescriptorSetIndex (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getDynamicProperty (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::getFunctionName (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getLanguage (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getNextResourceIndex (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::getNumDynamicPropertie (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::getPixelName (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getResourcePrefix (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getTextureBindingStart (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::getTextureMaxWidth (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::getUniqueID (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::GpuShaderCreator (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::hasDynamicProperty (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::operator= (C++ function), 374
OpenColorIO_v2_5::GpuShaderCreator::setAllowTexture1D (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::setDescriptorSetIndex (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::setFunctionName (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::setLanguage (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::setPixelName (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::setResourcePrefix (C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::setTextureMaxWidth (C++ function), 373
OpenColorIO_v2_5::GpuShaderCreator::setUniqueID

(C++ function), 372
OpenColorIO_v2_5::GpuShaderCreator::SizeGetter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureDimensions (C++ enum), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureDimensions::TEXTURE_1D (C++ enumerator), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureDimensions::TEXTURE_2D (C++ enumerator), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureType (C++ enum), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureType::TEXTURE_RED_CHANNEL (C++ enumerator), 371
OpenColorIO_v2_5::GpuShaderCreator::TextureType::TEXTURE_RGB_CHANNEL (C++ enumerator), 371
OpenColorIO_v2_5::GpuShaderCreator::VectorFloatGetter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreator::VectorIntGetter (C++ type), 371
OpenColorIO_v2_5::GpuShaderCreatorRcPtr (C++ type), 374
OpenColorIO_v2_5::GpuShaderDesc (C++ class), 382
OpenColorIO_v2_5::GpuShaderDesc::~GpuShaderDesc (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::clone (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::CreateShaderDesc (C++ function), 386
OpenColorIO_v2_5::GpuShaderDesc::get3DTexture (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::get3DTextureSamplingIndex (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::get3DTextureValues (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getNum3DTextures (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getNumTextures (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getNumUniforms (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getShaderText (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getTexture (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getTextureShaderBindingIndex (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getTextureValues (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getUniform (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::getUniformBufferSize (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::GpuShaderDesc (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::operator= (C++ function), 385
OpenColorIO_v2_5::GpuShaderDesc::UniformData (C++ struct), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_bufferOffset (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_getBool (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_getDouble (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_getFloat3 (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_type (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_vectorFloat (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::m_vectorInt (C++ member), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorFloat (C++ struct), 386
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorFloat::m_getVal (C++ member), 387
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorFloat::m_maxVal (C++ member), 387
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorInt (C++ struct), 387
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorInt::m_getVal (C++ member), 387
OpenColorIO_v2_5::GpuShaderDesc::UniformData::VectorInt::m_maxVal (C++ member), 387
OpenColorIO_v2_5::GpuShaderDescRcPtr (C++ type), 387
OpenColorIO_v2_5::GradingBSplineCurve (C++ class), 328
OpenColorIO_v2_5::GradingBSplineCurve::~GradingBSplineCurve (C++ function), 329
OpenColorIO_v2_5::GradingBSplineCurve::Create (C++ function), 329
OpenColorIO_v2_5::GradingBSplineCurve::createEditableCopy (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::getControlPoint (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::getNumControlPoints (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::getSlope (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::getSplineType (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::GradingBSplineCurve (C++ function), 329
OpenColorIO_v2_5::GradingBSplineCurve::operator= (C++ function), 329
OpenColorIO_v2_5::GradingBSplineCurve::setNumControlPoints

(C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::setSlope (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::setSplineType (C++ function), 329
OpenColorIO_v2_5::GradingBSplineCurve::slopesAreDefault (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurve::validate (C++ function), 328
OpenColorIO_v2_5::GradingBSplineCurveRcPtr (C++ type), 329
OpenColorIO_v2_5::GradingControlPoint (C++ struct), 327
OpenColorIO_v2_5::GradingControlPoint::GradingControlPoint (C++ function), 327
OpenColorIO_v2_5::GradingControlPoint::m_x (C++ member), 327
OpenColorIO_v2_5::GradingControlPoint::m_y (C++ member), 327
OpenColorIO_v2_5::GradingHueCurve (C++ class), 325
OpenColorIO_v2_5::GradingHueCurve::~GradingHueCurve (C++ function), 326
OpenColorIO_v2_5::GradingHueCurve::Create (C++ function), 326
OpenColorIO_v2_5::GradingHueCurve::createEditableCopy (C++ function), 325
OpenColorIO_v2_5::GradingHueCurve::GetBSplineTypeForHueCurveType (C++ function), 326
OpenColorIO_v2_5::GradingHueCurve::getCurve (C++ function), 326
OpenColorIO_v2_5::GradingHueCurve::getDrawCurveOnly (C++ function), 325
OpenColorIO_v2_5::GradingHueCurve::isIdentity (C++ function), 325
OpenColorIO_v2_5::GradingHueCurve::setDrawCurveOnly (C++ function), 326
OpenColorIO_v2_5::GradingHueCurve::validate (C++ function), 325
OpenColorIO_v2_5::GradingHueCurveRcPtr (C++ type), 326
OpenColorIO_v2_5::GradingHueCurveTransform (C++ class), 322
OpenColorIO_v2_5::GradingHueCurveTransform::~GradingHueCurveTransform (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::Create (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::equals (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::getFormatMetadata (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::getRGBToHSY (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::getSlope (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::getStyle (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::getTransformType (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::getValue (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::GradingHueCurveTransform (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::isDynamic (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::makeDynamic (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::makeNonDynamic (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::operator= (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::setRGBToHSY (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransform::setSlope (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::setStyle (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::setValue (C++ function), 323
OpenColorIO_v2_5::GradingHueCurveTransform::slopesAreDefault (C++ function), 324
OpenColorIO_v2_5::GradingHueCurveTransformRcPtr (C++ type), 324
OpenColorIO_v2_5::GradingPrimary (C++ struct), 314
OpenColorIO_v2_5::GradingPrimary::GradingPrimary (C++ function), 314
OpenColorIO_v2_5::GradingPrimary::m_brightness (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_clampBlack (C++ member), 315
OpenColorIO_v2_5::GradingPrimary::m_clampWhite (C++ member), 315
OpenColorIO_v2_5::GradingPrimary::m_contrast (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_exposure (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_gain (C++ member), 315
OpenColorIO_v2_5::GradingPrimary::m_gamma (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_lift (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_offset (C++ member), 314
OpenColorIO_v2_5::GradingPrimary::m_pivot (C++ member), 315
OpenColorIO_v2_5::GradingPrimary::m_pivotBlack

(*C++ member*), 315
OpenColorIO_v2_5::GradingPrimary::m_pivotWhite (*C++ member*), 315
OpenColorIO_v2_5::GradingPrimary::m_saturation (*C++ member*), 315
OpenColorIO_v2_5::GradingPrimary::NoClampBlack (*C++ function*), 315
OpenColorIO_v2_5::GradingPrimary::NoClampWhite (*C++ function*), 315
OpenColorIO_v2_5::GradingPrimary::validate (*C++ function*), 314
OpenColorIO_v2_5::GradingPrimaryTransform (*C++ class*), 312
OpenColorIO_v2_5::GradingPrimaryTransform::~GradingPrimaryTransform (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::Create (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::equals (*C++ function*), 312
OpenColorIO_v2_5::GradingPrimaryTransform::getFormatMetadata (*C++ function*), 312
OpenColorIO_v2_5::GradingPrimaryTransform::getStyle (*C++ function*), 312
OpenColorIO_v2_5::GradingPrimaryTransform::getTransformType (*C++ function*), 312
OpenColorIO_v2_5::GradingPrimaryTransform::getValue (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::GradingPrimaryTransform (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::isDynamic (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::makeDynamic (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::makeNonDynamic (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::operator= (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::setStyle (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransform::setValue (*C++ function*), 313
OpenColorIO_v2_5::GradingPrimaryTransformRcPtr (*C++ type*), 313
OpenColorIO_v2_5::GradingRGBCurve (*C++ class*), 320
OpenColorIO_v2_5::GradingRGBCurve::~GradingRGBCurve (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurve::Create (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurve::createEditableCopy (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurve::getCurve (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurve::isIdentity (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurve::validate (*C++ function*), 320
OpenColorIO_v2_5::GradingRGBCurveRcPtr (*C++ type*), 320
OpenColorIO_v2_5::GradingRGBCurveTransform (*C++ class*), 317
OpenColorIO_v2_5::GradingRGBCurveTransform::~GradingRGBCurveTransform (*C++ function*), 319
OpenColorIO_v2_5::GradingRGBCurveTransform::Create (*C++ function*), 319
OpenColorIO_v2_5::GradingRGBCurveTransform::equals (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getBypassLinToLog (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getFormatMetadata (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getSlope (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getStyle (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getTransformType (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::getValue (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::GradingRGBCurveTransform (*C++ function*), 319
OpenColorIO_v2_5::GradingRGBCurveTransform::isDynamic (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::makeDynamic (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::makeNonDynamic (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::operator= (*C++ function*), 319
OpenColorIO_v2_5::GradingRGBCurveTransform::setBypassLinToLog (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::setSlope (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::setStyle (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::setValue (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransform::slopesAreDefault (*C++ function*), 318
OpenColorIO_v2_5::GradingRGBCurveTransformRcPtr (*C++ type*), 319
OpenColorIO_v2_5::GradingRGBM (*C++ struct*), 315
OpenColorIO_v2_5::GradingRGBM::GradingRGBM (*C++ function*), 316
OpenColorIO_v2_5::GradingRGBM::m_blue (*C++ member*), 316
OpenColorIO_v2_5::GradingRGBM::m_green (*C++ member*), 316

OpenColorIO_v2_5::GradingRGBM::m_master (C++ member), 316
OpenColorIO_v2_5::GradingRGBM::m_red (C++ member), 316
OpenColorIO_v2_5::GradingRGBMSW (C++ struct), 333
OpenColorIO_v2_5::GradingRGBMSW::GradingRGBMSW (C++ function), 334
OpenColorIO_v2_5::GradingRGBMSW::m_blue (C++ member), 334
OpenColorIO_v2_5::GradingRGBMSW::m_green (C++ member), 334
OpenColorIO_v2_5::GradingRGBMSW::m_master (C++ member), 334
OpenColorIO_v2_5::GradingRGBMSW::m_red (C++ member), 334
OpenColorIO_v2_5::GradingRGBMSW::m_start (C++ member), 334
OpenColorIO_v2_5::GradingRGBMSW::m_width (C++ member), 334
OpenColorIO_v2_5::GradingStyle (C++ enum), 290
OpenColorIO_v2_5::GradingStyle::GRADING_LIN (C++ enumerator), 290
OpenColorIO_v2_5::GradingStyle::GRADING_LOG (C++ enumerator), 290
OpenColorIO_v2_5::GradingStyle::GRADING_VIDEO (C++ enumerator), 290
OpenColorIO_v2_5::GradingStyleFromString (C++ function), 310
OpenColorIO_v2_5::GradingStyleToString (C++ function), 310
OpenColorIO_v2_5::GradingTone (C++ struct), 332
OpenColorIO_v2_5::GradingTone::GradingTone (C++ function), 332
OpenColorIO_v2_5::GradingTone::m_blacks (C++ member), 333
OpenColorIO_v2_5::GradingTone::m_highlights (C++ member), 333
OpenColorIO_v2_5::GradingTone::m_midtones (C++ member), 333
OpenColorIO_v2_5::GradingTone::m_scontrast (C++ member), 333
OpenColorIO_v2_5::GradingTone::m_shadows (C++ member), 333
OpenColorIO_v2_5::GradingTone::m_whites (C++ member), 333
OpenColorIO_v2_5::GradingTone::validate (C++ function), 332
OpenColorIO_v2_5::GradingToneTransform (C++ class), 330
OpenColorIO_v2_5::GradingToneTransform::~GradingToneTransform (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::Create (C++ function), 332
OpenColorIO_v2_5::GradingToneTransform::equals (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::getFormatMetadata (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::getStyle (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::getTransformType (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::getValue (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::GradingToneTransfo (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::isDynamic (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::makeDynamic (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::makeNonDynamic (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::operator= (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::setStyle (C++ function), 331
OpenColorIO_v2_5::GradingToneTransform::setValue (C++ function), 331
OpenColorIO_v2_5::GradingToneTransformRcPtr (C++ type), 332
OpenColorIO_v2_5::GroupTransform (C++ class), 414
OpenColorIO_v2_5::GroupTransform::~GroupTransform (C++ function), 415
OpenColorIO_v2_5::GroupTransform::appendTransform (C++ function), 414
OpenColorIO_v2_5::GroupTransform::Create (C++ function), 415
OpenColorIO_v2_5::GroupTransform::GetFormatExtensionByInde (C++ function), 415
OpenColorIO_v2_5::GroupTransform::getFormatMetadata (C++ function), 414
OpenColorIO_v2_5::GroupTransform::GetFormatNameByIndex (C++ function), 415
OpenColorIO_v2_5::GroupTransform::getNumTransforms (C++ function), 414
OpenColorIO_v2_5::GroupTransform::GetNumWriteFormats (C++ function), 415
OpenColorIO_v2_5::GroupTransform::getTransform (C++ function), 414
OpenColorIO_v2_5::GroupTransform::GroupTransform (C++ function), 415
OpenColorIO_v2_5::GroupTransform::operator= (C++ function), 415
OpenColorIO_v2_5::GroupTransform::prependTransform (C++ function), 414
OpenColorIO_v2_5::GroupTransform::write (C++ function), 414

OpenColorIO_v2_5::GroupTransformRcPtr (*C++ type*), 415
OpenColorIO_v2_5::ImageDesc (*C++ class*), 335
OpenColorIO_v2_5::ImageDesc::~ImageDesc (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getAData (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getBData (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getBitDepth (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getGData (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getHeight (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getRData (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getWidth (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getXStrideBytes (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::getYStrideBytes (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::ImageDesc (*C++ function*), 335
OpenColorIO_v2_5::ImageDesc::isFloat (*C++ function*), 336
OpenColorIO_v2_5::ImageDesc::isRGBAPacked (*C++ function*), 336
OpenColorIO_v2_5::ImageDescRcPtr (*C++ type*), 336
OpenColorIO_v2_5::Interpolation (*C++ enum*), 276
OpenColorIO_v2_5::Interpolation::INTERP_BEST (*C++ enumerator*), 277
OpenColorIO_v2_5::Interpolation::INTERP_CUBIC (*C++ enumerator*), 277
OpenColorIO_v2_5::Interpolation::INTERP_DEFAULT (*C++ enumerator*), 277
OpenColorIO_v2_5::Interpolation::INTERP_LINEAR (*C++ enumerator*), 276
OpenColorIO_v2_5::Interpolation::INTERP_NEAREST (*C++ enumerator*), 276
OpenColorIO_v2_5::Interpolation::INTERP_TETRAHEDRAL (*C++ enumerator*), 277
OpenColorIO_v2_5::Interpolation::INTERP_UNKNOWN (*C++ enumerator*), 276
OpenColorIO_v2_5::InterpolationFromString (*C++ function*), 309
OpenColorIO_v2_5::InterpolationToString (*C++ function*), 309
OpenColorIO_v2_5::IsEnvVariablePresent (*C++ function*), 306
OpenColorIO_v2_5::LegacyViewingPipeline (*C++ class*), 188
OpenColorIO_v2_5::LegacyViewingPipeline::~LegacyViewingPip (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::Create (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::getChannelView (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getColorTimingCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getDisplayCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getDisplayViewTra (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getLinearCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getLooksOverride (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::getLooksOverrideE (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::getProcessor (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::LegacyViewingPipe (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::operator= (*C++ function*), 190
OpenColorIO_v2_5::LegacyViewingPipeline::setChannelView (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setColorTimingCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setDisplayCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setDisplayViewTra (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setLinearCC (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setLooksOverride (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipeline::setLooksOverrideE (*C++ function*), 189
OpenColorIO_v2_5::LegacyViewingPipelineRcPtr (*C++ type*), 190
OpenColorIO_v2_5::LogAffineTransform (*C++ class*), 416
OpenColorIO_v2_5::LogAffineTransform::~LogAffineTransform (*C++ function*), 417
OpenColorIO_v2_5::LogAffineTransform::Create (*C++ function*), 417
OpenColorIO_v2_5::LogAffineTransform::equals (*C++ function*), 417
OpenColorIO_v2_5::LogAffineTransform::getBase (*C++ function*), 417
OpenColorIO_v2_5::LogAffineTransform::getFormatMetadata (*C++ function*), 417
OpenColorIO_v2_5::LogAffineTransform::getLinSideOffsetValu

(C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::getLinSideOffsetValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::getLogSideOffsetValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::getLogSideSlopeValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::getTransformType (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::LogAffineTransform (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::operator= (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::setBase (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::setLinSideOffsetValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::setLinSideSlopeValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::setLogSideOffsetValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransform::setLogSideSlopeValue (C++ function), 417
OpenColorIO_v2_5::LogAffineTransformRcPtr (C++ type), 417
OpenColorIO_v2_5::LogCameraTransform (C++ class), 419
OpenColorIO_v2_5::LogCameraTransform::~LogCameraTransform (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::Create (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::equals (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getBase (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getFormatMetadata (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getLinearSlopeValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::getLinSideBreakValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::getLinSideOffsetValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::getLinSideSlopeValue (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getLogSideOffsetValue (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getLogSideSlopeValue (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::getTransformType (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::LogCameraTransform (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::operator= (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::setBase (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::setLinearSlopeValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::setLinSideBreakValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::setLinSideOffsetValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::setLinSideSlopeValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransform::setLogSideOffsetValue (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::setLogSideSlopeValue (C++ function), 419
OpenColorIO_v2_5::LogCameraTransform::unsetLinearSlopeValue (C++ function), 420
OpenColorIO_v2_5::LogCameraTransformRcPtr (C++ type), 420
OpenColorIO_v2_5::LoggingFunction (C++ type), 305
OpenColorIO_v2_5::LoggingLevel (C++ enum), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_DEBUG (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_DEFAULT (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_INFO (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_NONE (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_UNKNOWN (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevel::LOGGING_LEVEL_WARNING (C++ enumerator), 268
OpenColorIO_v2_5::LoggingLevelFromString (C++ function), 307
OpenColorIO_v2_5::LoggingLevelToString (C++ function), 307
OpenColorIO_v2_5::LogMessage (C++ function), 305
OpenColorIO_v2_5::LogTransform (C++ class), 421
OpenColorIO_v2_5::LogTransform::~LogTransform (C++ function), 421
OpenColorIO_v2_5::LogTransform::Create (C++ function), 422
OpenColorIO_v2_5::LogTransform::equals (C++ function), 421
OpenColorIO_v2_5::LogTransform::getBase (C++ function), 421
OpenColorIO_v2_5::LogTransform::getFormatMetadata (C++ function), 421
OpenColorIO_v2_5::LogTransform::getTransformType (C++ function), 421
OpenColorIO_v2_5::LogTransform::LogTransform (C++ function), 421

OpenColorIO_v2_5::LogTransform::operator= (C++ function), 421
OpenColorIO_v2_5::LogTransform::setBase (C++ function), 421
OpenColorIO_v2_5::LogTransformRcPtr (C++ type), 422
OpenColorIO_v2_5::Look (C++ class), 342
OpenColorIO_v2_5::Look::~Look (C++ function), 343
OpenColorIO_v2_5::Look::Create (C++ function), 343
OpenColorIO_v2_5::Look::createEditableCopy (C++ function), 342
OpenColorIO_v2_5::Look::getDescription (C++ function), 342
OpenColorIO_v2_5::Look::getInterchangeAttribute (C++ function), 342
OpenColorIO_v2_5::Look::getInterchangeAttributes (C++ function), 343
OpenColorIO_v2_5::Look::getInverseTransform (C++ function), 342
OpenColorIO_v2_5::Look::getName (C++ function), 342
OpenColorIO_v2_5::Look::getProcessSpace (C++ function), 342
OpenColorIO_v2_5::Look::getTransform (C++ function), 342
OpenColorIO_v2_5::Look::Look (C++ function), 343
OpenColorIO_v2_5::Look::operator= (C++ function), 343
OpenColorIO_v2_5::Look::setDescription (C++ function), 342
OpenColorIO_v2_5::Look::setInterchangeAttribute (C++ function), 342
OpenColorIO_v2_5::Look::setInverseTransform (C++ function), 342
OpenColorIO_v2_5::Look::setName (C++ function), 342
OpenColorIO_v2_5::Look::setProcessSpace (C++ function), 342
OpenColorIO_v2_5::Look::setTransform (C++ function), 342
OpenColorIO_v2_5::LookRcPtr (C++ type), 343
OpenColorIO_v2_5::LookTransform (C++ class), 423
OpenColorIO_v2_5::LookTransform::~LookTransform (C++ function), 423
OpenColorIO_v2_5::LookTransform::Create (C++ function), 423
OpenColorIO_v2_5::LookTransform::createEditableCopy (C++ function), 423
OpenColorIO_v2_5::LookTransform::getDirection (C++ function), 423
OpenColorIO_v2_5::LookTransform::getDst (C++ function), 423
OpenColorIO_v2_5::LookTransform::getLooks (C++ function), 423
OpenColorIO_v2_5::LookTransform::GetLooksResultColorSpace (C++ function), 423
OpenColorIO_v2_5::LookTransform::getSkipColorSpaceConversi (C++ function), 423
OpenColorIO_v2_5::LookTransform::getSrc (C++ function), 423
OpenColorIO_v2_5::LookTransform::getTransformType (C++ function), 423
OpenColorIO_v2_5::LookTransform::operator= (C++ function), 423
OpenColorIO_v2_5::LookTransform::setDirection (C++ function), 423
OpenColorIO_v2_5::LookTransform::setDst (C++ function), 423
OpenColorIO_v2_5::LookTransform::setLooks (C++ function), 423
OpenColorIO_v2_5::LookTransform::setSkipColorSpaceConversi (C++ function), 423
OpenColorIO_v2_5::LookTransform::setSrc (C++ function), 423
OpenColorIO_v2_5::LookTransform::validate (C++ function), 423
OpenColorIO_v2_5::LookTransformRcPtr (C++ type), 424
OpenColorIO_v2_5::Lut1DHueAdjust (C++ enum), 279
OpenColorIO_v2_5::Lut1DHueAdjust::HUE_DW3 (C++ enumerator), 279
OpenColorIO_v2_5::Lut1DHueAdjust::HUE_NONE (C++ enumerator), 279
OpenColorIO_v2_5::Lut1DHueAdjust::HUE_WYPN (C++ enumerator), 279
OpenColorIO_v2_5::Lut1DTransform (C++ class), 425
OpenColorIO_v2_5::Lut1DTransform::~Lut1DTransform (C++ function), 427
OpenColorIO_v2_5::Lut1DTransform::Create (C++ function), 427
OpenColorIO_v2_5::Lut1DTransform::equals (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getFileOutputBitDepth (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getFormatMetadata (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getHueAdjust (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getInputHalfDomain (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getInterpolation (C++ function), 426
OpenColorIO_v2_5::Lut1DTransform::getLength

(*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::getOutputRawHalfs (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::getTransformType (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::getValue (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::Lut1DTransform (*C++ function*), 427
OpenColorIO_v2_5::Lut1DTransform::operator= (*C++ function*), 427
OpenColorIO_v2_5::Lut1DTransform::setFileOutputBitDepth (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setHueAdjust (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setInputHalfsEmulation (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setInterpolation (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setLength (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setOutputRawHalfs (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransform::setValue (*C++ function*), 426
OpenColorIO_v2_5::Lut1DTransformRcPtr (*C++ type*), 427
OpenColorIO_v2_5::Lut3DTransform (*C++ class*), 428
OpenColorIO_v2_5::Lut3DTransform::~Lut3DTransform (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::Create (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::equals (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::getFileOutputBitDepth (*C++ function*), 428
OpenColorIO_v2_5::Lut3DTransform::getFormatMetadata (*C++ function*), 428, 429
OpenColorIO_v2_5::Lut3DTransform::getGridSize (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::getInterpolation (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::getTransformType (*C++ function*), 428
OpenColorIO_v2_5::Lut3DTransform::getValue (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::Lut3DTransform (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::operator= (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::setFileOutputBitDepth (*C++ function*), 428
OpenColorIO_v2_5::Lut3DTransform::setGridSize (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::setInterpolation (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransform::setValue (*C++ function*), 429
OpenColorIO_v2_5::Lut3DTransformRcPtr (*C++ type*), 429
OpenColorIO_v2_5::MatrixTransform (*C++ class*), 431
OpenColorIO_v2_5::MatrixTransform::~MatrixTransform (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::Create (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::equals (*C++ function*), 431
OpenColorIO_v2_5::MatrixTransform::Fit (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::getFileInputBitDepth (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::getFileOutputBitDepth (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::getFormatMetadata (*C++ function*), 431
OpenColorIO_v2_5::MatrixTransform::getMatrix (*C++ function*), 431
OpenColorIO_v2_5::MatrixTransform::getOffset (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::getTransformType (*C++ function*), 431
OpenColorIO_v2_5::MatrixTransform::Identity (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::MatrixTransform (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::operator= (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::Sat (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::Scale (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::setFileInputBitDepth (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::setFileOutputBitDepth (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::setMatrix (*C++ function*), 431
OpenColorIO_v2_5::MatrixTransform::setOffset (*C++ function*), 432
OpenColorIO_v2_5::MatrixTransform::View (*C++ function*), 433
OpenColorIO_v2_5::MatrixTransformRcPtr (*C++ type*), 433
OpenColorIO_v2_5::MixingColorSpaceManager (*C++ class*), 193
OpenColorIO_v2_5::MixingColorSpaceManager::~MixingColorSpa

(C++ function), 194
OpenColorIO_v2_5::MixingColorSpaceManager::Create (C++ function), 194
OpenColorIO_v2_5::MixingColorSpaceManager::getMixingEncoding (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getMixingSpaceName (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getNumMixingEncodings (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getNumMixingSpaces (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getProcessor (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getSelectedMixingEncodingIdx (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getSelectedMixingSpaceIdx (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::getSlider (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::isPerceptuallyUniform (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::MixingColorSpaceManager (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::operator= (C++ function), 194
OpenColorIO_v2_5::MixingColorSpaceManager::refresh (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::setSelectedMixingEncoding (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::setSelectedMixingEncodingIdx (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::setSelectedMixingSpace (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManager::setSelectedMixingSpaceIdx (C++ function), 193
OpenColorIO_v2_5::MixingColorSpaceManagerRcPtr (C++ type), 194
OpenColorIO_v2_5::MixingSlider (C++ class), 192
OpenColorIO_v2_5::MixingSlider::~MixingSlider (C++ function), 193
OpenColorIO_v2_5::MixingSlider::getSliderMaxEdge (C++ function), 192
OpenColorIO_v2_5::MixingSlider::getSliderMinEdge (C++ function), 192
OpenColorIO_v2_5::MixingSlider::MixingSlider (C++ function), 193
OpenColorIO_v2_5::MixingSlider::mixingToSlider (C++ function), 193
OpenColorIO_v2_5::MixingSlider::operator= (C++ function), 193
OpenColorIO_v2_5::MixingSlider::setSliderMaxEdge (C++ function), 192
OpenColorIO_v2_5::MixingSlider::setSliderMinEdge (C++ function), 192
OpenColorIO_v2_5::MixingSlider::sliderToMixing (C++ function), 193
OpenColorIO_v2_5::NamedTransform (C++ class), 344
OpenColorIO_v2_5::NamedTransform::~NamedTransform (C++ function), 346
OpenColorIO_v2_5::NamedTransform::addAlias (C++ function), 345
OpenColorIO_v2_5::NamedTransform::addCategory (C++ function), 345
OpenColorIO_v2_5::NamedTransform::clearAliases (C++ function), 345
OpenColorIO_v2_5::NamedTransform::clearCategories (C++ function), 345
OpenColorIO_v2_5::NamedTransform::Create (C++ function), 346
OpenColorIO_v2_5::NamedTransform::createEditableCopy (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getAlias (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getCategory (C++ function), 346
OpenColorIO_v2_5::NamedTransform::getDescription (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getEncoding (C++ function), 346
OpenColorIO_v2_5::NamedTransform::getFamily (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getName (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getNumAliases (C++ function), 345
OpenColorIO_v2_5::NamedTransform::getNumCategories (C++ function), 346
OpenColorIO_v2_5::NamedTransform::GetTransform (C++ function), 346
OpenColorIO_v2_5::NamedTransform::getTransform (C++ function), 346
OpenColorIO_v2_5::NamedTransform::hasAlias (C++ function), 345
OpenColorIO_v2_5::NamedTransform::hasCategory (C++ function), 345
OpenColorIO_v2_5::NamedTransform::NamedTransform (C++ function), 346
OpenColorIO_v2_5::NamedTransform::operator= (C++ function), 346
OpenColorIO_v2_5::NamedTransform::removeAlias (C++ function), 345
OpenColorIO_v2_5::NamedTransform::removeCategory (C++ function), 345
OpenColorIO_v2_5::NamedTransform::setDescription (C++ function), 345
OpenColorIO_v2_5::NamedTransform::setEncoding (C++ function), 346

OpenColorIO_v2_5::NamedTransform::setFamily (C++ function), 345
OpenColorIO_v2_5::NamedTransform::setName (C++ function), 345
OpenColorIO_v2_5::NamedTransform::setTransform (C++ function), 346
OpenColorIO_v2_5::NamedTransformRcPtr (C++ type), 346
OpenColorIO_v2_5::NamedTransformVisibility (C++ enum), 270
OpenColorIO_v2_5::NamedTransformVisibility::NAMEDTRANSFORM_ACTIVE (C++ enumerator), 270
OpenColorIO_v2_5::NamedTransformVisibility::NAMEDTRANSFORM_ALL (C++ enumerator), 270
OpenColorIO_v2_5::NamedTransformVisibility::NAMEDTRANSFORM_INACTIVE (C++ enumerator), 270
OpenColorIO_v2_5::NegativeStyle (C++ enum), 289
OpenColorIO_v2_5::NegativeStyle::NEGATIVE_CLAMP (C++ enumerator), 289
OpenColorIO_v2_5::NegativeStyle::NEGATIVE_LINEAR (C++ enumerator), 289
OpenColorIO_v2_5::NegativeStyle::NEGATIVE_MIRROR (C++ enumerator), 289
OpenColorIO_v2_5::NegativeStyle::NEGATIVE_PASS_THRU (C++ enumerator), 289
OpenColorIO_v2_5::NegativeStyleFromString (C++ function), 311
OpenColorIO_v2_5::NegativeStyleToString (C++ function), 311
OpenColorIO_v2_5::operator<< (C++ function), 184, 186, 190, 193, 194, 205, 253, 261, 303, 313, 315, 316, 319, 320, 324, 326, 327, 329, 332–334, 336, 343, 346, 362, 366, 389, 391, 392, 397, 398, 400, 402, 404, 408, 411, 412, 415, 417, 420, 422, 423, 427, 429, 433, 437, 440
OpenColorIO_v2_5::OptimizationFlags (C++ enum), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_ALL (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_EXPONENT (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_GAMMA (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_LUT1D (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_LUT3D (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_MATRIX (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_RANGE (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_COMP_SOP (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_DEFAULT (C++ enumerator), 298
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_DRAFT (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_FAST_LOG (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_GOOD (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_IDENTITY (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_IDENTITY (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_LOSSLESS (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_LUT_INV_ (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_NO_DYNAM (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_NONE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_PAIR_IDE (C++ enumerator), 296
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_SIMPLIFY (C++ enumerator), 297
OpenColorIO_v2_5::OptimizationFlags::OPTIMIZATION_VERY_GOO (C++ enumerator), 297
OpenColorIO_v2_5::PackedImageDesc (C++ class), 337
OpenColorIO_v2_5::PackedImageDesc::~PackedImageDesc (C++ function), 338
OpenColorIO_v2_5::PackedImageDesc::getADataOpenColorIO_v2_5::PackedImageDesc::getBDataOpenColorIO_v2_5::PackedImageDesc::getBitDepthOpenColorIO_v2_5::PackedImageDesc::getChannelOrderOpenColorIO_v2_5::PackedImageDesc::getChanStrideBytes

(*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getData (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getGData (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getHeight (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getNumChannels (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getRData (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getWidth (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getXStrideBytes (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::getYStrideBytes (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::isFloat (*C++ function*), 339
OpenColorIO_v2_5::PackedImageDesc::isRGBAPacked (*C++ function*), 338
OpenColorIO_v2_5::PackedImageDesc::PackedImageDesc (*C++ function*), 338
OpenColorIO_v2_5::PlanarImageDesc (*C++ class*), 340
OpenColorIO_v2_5::PlanarImageDesc::~PlanarImageDesc (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getAData (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getBData (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getBitDepth (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getGData (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getHeight (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::getRData (*C++ function*), 340
OpenColorIO_v2_5::PlanarImageDesc::getWidth (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::getXStrideBytes (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::getYStrideBytes (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::isFloat (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::isRGBAPacked (*C++ function*), 341
OpenColorIO_v2_5::PlanarImageDesc::PlanarImageDesc (*C++ function*), 340
OpenColorIO_v2_5::Processor (*C++ class*), 349
OpenColorIO_v2_5::Processor::~Processor (*C++ function*), 351
OpenColorIO_v2_5::Processor::createGroupTransform (*C++ function*), 349
OpenColorIO_v2_5::Processor::getCacheID (*C++ function*), 349
OpenColorIO_v2_5::Processor::getDefaultCPUProcessor (*C++ function*), 350
OpenColorIO_v2_5::Processor::getDefaultGPUProcessor (*C++ function*), 350
OpenColorIO_v2_5::Processor::getDynamicProperty (*C++ function*), 349
OpenColorIO_v2_5::Processor::getFormatMetadata (*C++ function*), 349
OpenColorIO_v2_5::Processor::getNumTransforms (*C++ function*), 349
OpenColorIO_v2_5::Processor::getOptimizedCPUProcessor (*C++ function*), 351
OpenColorIO_v2_5::Processor::getOptimizedGPUProcessor (*C++ function*), 350
OpenColorIO_v2_5::Processor::getOptimizedLegacyGPUProcesso (*C++ function*), 350
OpenColorIO_v2_5::Processor::getOptimizedProcessor (*C++ function*), 350
OpenColorIO_v2_5::Processor::getProcessorMetadata (*C++ function*), 349
OpenColorIO_v2_5::Processor::getTransformFormatMetadata (*C++ function*), 349
OpenColorIO_v2_5::Processor::hasChannelCrosstalk (*C++ function*), 349
OpenColorIO_v2_5::Processor::hasDynamicProperty (*C++ function*), 350
OpenColorIO_v2_5::Processor::isDynamic (*C++ function*), 350
OpenColorIO_v2_5::Processor::isNoOp (*C++ function*), 349
OpenColorIO_v2_5::Processor::operator= (*C++ function*), 351
OpenColorIO_v2_5::Processor::Processor (*C++ function*), 351
OpenColorIO_v2_5::ProcessorCacheFlags (*C++ enum*), 298
OpenColorIO_v2_5::ProcessorCacheFlags::PROCESSOR_CACHE_DEF (*C++ enumerator*), 298
OpenColorIO_v2_5::ProcessorCacheFlags::PROCESSOR_CACHE_ENA (*C++ enumerator*), 298
OpenColorIO_v2_5::ProcessorCacheFlags::PROCESSOR_CACHE_OFF (*C++ enumerator*), 298
OpenColorIO_v2_5::ProcessorCacheFlags::PROCESSOR_CACHE_SHA (*C++ enumerator*), 298
OpenColorIO_v2_5::ProcessorMetadata (*C++ class*), 356
OpenColorIO_v2_5::ProcessorMetadata::~ProcessorMetadata (*C++ function*), 357
OpenColorIO_v2_5::ProcessorMetadata::addFile (*C++ function*), 356

OpenColorIO_v2_5::ProcessorMetadata::addLook (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadata::Create (C++ function), 357
OpenColorIO_v2_5::ProcessorMetadata::getFile (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadata::getLook (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadata::getNumFiles (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadata::getNumLooks (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadata::operator= (C++ function), 357
OpenColorIO_v2_5::ProcessorMetadata::ProcessorMetadata (C++ function), 356
OpenColorIO_v2_5::ProcessorMetadataRcPtr (C++ type), 357
OpenColorIO_v2_5::ProcessorRcPtr (C++ type), 351
OpenColorIO_v2_5::RangeStyle (C++ enum), 283
OpenColorIO_v2_5::RangeStyle::RANGE_CLAMP (C++ enumerator), 283
OpenColorIO_v2_5::RangeStyle::RANGE_NO_CLAMP (C++ enumerator), 283
OpenColorIO_v2_5::RangeStyleFromString (C++ function), 310
OpenColorIO_v2_5::RangeStyleToString (C++ function), 310
OpenColorIO_v2_5::RangeTransform (C++ class), 435
OpenColorIO_v2_5::RangeTransform::~RangeTransform (C++ function), 437
OpenColorIO_v2_5::RangeTransform::Create (C++ function), 437
OpenColorIO_v2_5::RangeTransform::equals (C++ function), 435
OpenColorIO_v2_5::RangeTransform::getFileInputBitDepth (C++ function), 436
OpenColorIO_v2_5::RangeTransform::getFileOutputBitDepth (C++ function), 436
OpenColorIO_v2_5::RangeTransform::getFormatMetadata (C++ function), 435
OpenColorIO_v2_5::RangeTransform::getMaxInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::getMaxOutValue (C++ function), 437
OpenColorIO_v2_5::RangeTransform::getMinInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::getMinOutValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::getStyle (C++ function), 435
OpenColorIO_v2_5::RangeTransform::getTransformType (C++ function), 435
OpenColorIO_v2_5::RangeTransform::hasMaxInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::hasMaxOutValue (C++ function), 437
OpenColorIO_v2_5::RangeTransform::hasMinInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::hasMinOutValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::operator= (C++ function), 437
OpenColorIO_v2_5::RangeTransform::RangeTransform (C++ function), 437
OpenColorIO_v2_5::RangeTransform::setFileInputBitDepth (C++ function), 436
OpenColorIO_v2_5::RangeTransform::setFileOutputBitDepth (C++ function), 436
OpenColorIO_v2_5::RangeTransform::setMaxInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::setMaxOutValue (C++ function), 437
OpenColorIO_v2_5::RangeTransform::setMinInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::setMinOutValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::setStyle (C++ function), 435
OpenColorIO_v2_5::RangeTransform::unsetMaxInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::unsetMaxOutValue (C++ function), 437
OpenColorIO_v2_5::RangeTransform::unsetMinInValue (C++ function), 436
OpenColorIO_v2_5::RangeTransform::unsetMinOutValue (C++ function), 436
OpenColorIO_v2_5::RangeTransformRcPtr (C++ type), 437
OpenColorIO_v2_5::ReferenceSpaceType (C++ enum), 268
OpenColorIO_v2_5::ReferenceSpaceType::REFERENCE_SPACE_DISP (C++ enumerator), 269
OpenColorIO_v2_5::ReferenceSpaceType::REFERENCE_SPACE_SCEN (C++ enumerator), 268
OpenColorIO_v2_5::ResetComputeHashFunction (C++ function), 306
OpenColorIO_v2_5::ResetToDefaultLoggingFunction (C++ function), 305
OpenColorIO_v2_5::RGBCurveType (C++ enum), 292
OpenColorIO_v2_5::RGBCurveType::RGB_BLUE (C++ enumerator), 292
OpenColorIO_v2_5::RGBCurveType::RGB_GREEN (C++ enumerator), 292
OpenColorIO_v2_5::RGBCurveType::RGB_MASTER (C++ enumerator), 293

OpenColorIO_v2_5::RGBCurveType::RGB_NUM_CURVES (C++ enumerator), 293
OpenColorIO_v2_5::RGBCurveType::RGB_RED (C++ enumerator), 292
OpenColorIO_v2_5::SearchReferenceSpaceType (C++ enum), 269
OpenColorIO_v2_5::SearchReferenceSpaceType::SEARCH_REFERENCE_SPACE_ALL (C++ enumerator), 269
OpenColorIO_v2_5::SearchReferenceSpaceType::SEARCH_REFERENCE_SPACE_DISPLAY (C++ enumerator), 269
OpenColorIO_v2_5::SearchReferenceSpaceType::SEARCH_REFERENCE_SPACE_SCENE (C++ enumerator), 269
OpenColorIO_v2_5::SetComputeHashFunction (C++ function), 306
OpenColorIO_v2_5::SetCurrentConfig (C++ function), 232
OpenColorIO_v2_5::SetEnvVariable (C++ function), 306
OpenColorIO_v2_5::SetLoggingFunction (C++ function), 305
OpenColorIO_v2_5::SetLoggingLevel (C++ function), 305
OpenColorIO_v2_5::SystemMonitors (C++ class), 387
OpenColorIO_v2_5::SystemMonitors::Get (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::getMonitorName (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::getNumMonitors (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::getProfileFilepath (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::isSupported (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::operator= (C++ function), 388
OpenColorIO_v2_5::SystemMonitors::SystemMonitors (C++ function), 388
OpenColorIO_v2_5::SystemMonitorsRcPtr (C++ type), 388
OpenColorIO_v2_5::Transform (C++ class), 389
OpenColorIO_v2_5::Transform::~Transform (C++ function), 389
OpenColorIO_v2_5::Transform::createEditableCopy (C++ function), 389
OpenColorIO_v2_5::Transform::getDirection (C++ function), 389
OpenColorIO_v2_5::Transform::getTransformType (C++ function), 389
OpenColorIO_v2_5::Transform::operator= (C++ function), 389
OpenColorIO_v2_5::Transform::setDirection (C++ function), 389
OpenColorIO_v2_5::Transform::Transform (C++ function), 389
OpenColorIO_v2_5::Transform::validate (C++ function), 389
OpenColorIO_v2_5::TransformDirection (C++ enum), 272
OpenColorIO_v2_5::TransformDirection::TRANSFORM_DIR_FORWARD (C++ enumerator), 272
OpenColorIO_v2_5::TransformDirection::TRANSFORM_DIR_INVERSE (C++ enumerator), 272
OpenColorIO_v2_5::TransformDirectionFromString (C++ function), 308
OpenColorIO_v2_5::TransformDirectionToString (C++ function), 308
OpenColorIO_v2_5::TransformRcPtr (C++ type), 389
OpenColorIO_v2_5::TransformType (C++ enum), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_ALLOCATION (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_BUILTIN (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_CDL (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_COLORSPACE (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_DISPLAY_VI (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_EXPONENT (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_EXPONENT_W (C++ enumerator), 274
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_EXPOSURE_C (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_FILE (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_FIXED_FUNC (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_GRADING_HU (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_GRADING_PR (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_GRADING_RC (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_GRADING_TO (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_GROUP (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LOG (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LOG_AFFINE (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LOG_CAMERA (C++ enumerator), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LOOK

(C++ *enumerator*), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LUT1D (C++ *enumerator*), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_LUT3D (C++ *enumerator*), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_MATRIX (C++ *enumerator*), 275
OpenColorIO_v2_5::TransformType::TRANSFORM_TYPE_RANGE (C++ *enumerator*), 275
OpenColorIO_v2_5::UniformDataType (C++ *enum*), 293
OpenColorIO_v2_5::UniformDataType::UNIFORM_BOOL (C++ *enumerator*), 293
OpenColorIO_v2_5::UniformDataType::UNIFORM_DOUBLE (C++ *enumerator*), 293
OpenColorIO_v2_5::UniformDataType::UNIFORM_FLOAT3 (C++ *enumerator*), 293
OpenColorIO_v2_5::UniformDataType::UNIFORM_UNKNOWN (C++ *enumerator*), 294
OpenColorIO_v2_5::UniformDataType::UNIFORM_VECTOR_FLOAT4 (C++ *enumerator*), 293
OpenColorIO_v2_5::UniformDataType::UNIFORM_VECTOR_INT (C++ *enumerator*), 294
OpenColorIO_v2_5::UnsetEnvVariable (C++ *function*), 306
OpenColorIO_v2_5::ViewingRules (C++ *class*), 364
OpenColorIO_v2_5::ViewingRules::~ViewingRules (C++ *function*), 366
OpenColorIO_v2_5::ViewingRules::addColorSpace (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::addEncoding (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::Create (C++ *function*), 366
OpenColorIO_v2_5::ViewingRules::createEditableCopy (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getColorSpace (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getCustomKeyName (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getCustomKeyValue (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getEncoding (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getIndexForRule (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getName (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getNumColorSpaces (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getNumCustomKeys (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getNumEncodings (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::getNumEntries (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::insertRule (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::operator= (C++ *function*), 366
OpenColorIO_v2_5::ViewingRules::removeColorSpace (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::removeEncoding (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::removeRule (C++ *function*), 366
OpenColorIO_v2_5::ViewingRules::setCustomKey (C++ *function*), 365
OpenColorIO_v2_5::ViewingRules::ViewingRules (C++ *function*), 366
OpenColorIO_v2_5::ViewingRulesRcPtr (C++ *type*), 366
OpenColorIO_v2_5::ViewTransform (C++ *class*), 439
OpenColorIO_v2_5::ViewTransform::~ViewTransform (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::addCategory (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::clearCategories (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::Create (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::createEditableCopy (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getCategory (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::getDescription (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getFamily (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getInterchangeAttribute (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getInterchangeAttributes (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getName (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::getNumCategories (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::getReferenceSpaceType (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::getTransform (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::hasCategory (C++ *function*), 439
OpenColorIO_v2_5::ViewTransform::operator= (C++ *function*), 440
OpenColorIO_v2_5::ViewTransform::removeCategory (C++ *function*), 440

OpenColorIO_v2_5::ViewTransform::setDescription (*C++ function*), 439
OpenColorIO_v2_5::ViewTransform::setFamily (*C++ function*), 439
OpenColorIO_v2_5::ViewTransform::setInterchangeAttribute (*C++ function*), 439
OpenColorIO_v2_5::ViewTransform::setName (*C++ function*), 439
OpenColorIO_v2_5::ViewTransform::setTransform (*C++ function*), 440
OpenColorIO_v2_5::ViewTransform::ViewTransform (*C++ function*), 440
OpenColorIO_v2_5::ViewTransformDirection (*C++ enum*), 272
OpenColorIO_v2_5::ViewTransformDirection::VIEWTRANSFORM_DIR_FROM_REFERENCE (*C++ enumerator*), 272
OpenColorIO_v2_5::ViewTransformDirection::VIEWTRANSFORM_DIR_TO_REFERENCE (*C++ enumerator*), 272
OpenColorIO_v2_5::ViewTransformRcPtr (*C++ type*), 440
OpenColorIO_v2_5::ViewType (*C++ enum*), 271
OpenColorIO_v2_5::ViewType::VIEW_DISPLAY_DEFINED (*C++ enumerator*), 271
OpenColorIO_v2_5::ViewType::VIEW_SHARED (*C++ enumerator*), 271
OPTIMIZATION_ALL (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_EXPONENT (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_GAMMA (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_LUT1D (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_LUT3D (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_MATRIX (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_RANGE (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_COMP_SEPARABLE_PREFIX (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_DEFAULT (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_DRAFT (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_FAST_LOG_EXP_POW (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_GOOD (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_IDENTITY (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_IDENTITY_GAMMA (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_LOSSLESS (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_LUT_INV_FAST (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_NO_DYNAMIC_PROPERTIES (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_NONE (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_PAIR_IDENTITY_CDL (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_PAIR_IDENTITY_EXPOSURE_CONTRAST (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_PAIR_IDENTITY_FIXED_FUNCTION (*PyOpenColorIO.OptimizationFlags attribute*), 295
OPTIMIZATION_PAIR_IDENTITY_GAMMA (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_PAIR_IDENTITY_GRADING (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_PAIR_IDENTITY_LOG (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_PAIR_IDENTITY_LUT1D (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_PAIR_IDENTITY_LUT3D (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_SIMPLIFY_OPS (*PyOpenColorIO.OptimizationFlags attribute*), 296
OPTIMIZATION_VERY_GOOD (*PyOpenColorIO.OptimizationFlags attribute*), 296
OptimizationFlags (*class in PyOpenColorIO*), 294

P

PackedImageDesc (*class in PyOpenColorIO*), 336
parseColorSpaceFromString() (*PyOpenColorIO.Config method*), 226
pivot (*PyOpenColorIO.GradingPrimary property*), 314
pivotBlack (*PyOpenColorIO.GradingPrimary property*), 314
pivotWhite (*PyOpenColorIO.GradingPrimary property*), 314
PlanarImageDesc (*class in PyOpenColorIO*), 339
prependTransform() (*PyOpenColorIO.GroupTransform method*), 413
Processor (*class in PyOpenColorIO*), 347
PROCESSOR_CACHE_DEFAULT (*PyOpenColorIO.ProcessorCacheFlags attribute*), 298
PROCESSOR_CACHE_ENABLED (*PyOpenColorIO.ProcessorCacheFlags attribute*), 298
PROCESSOR_CACHE_OFF (*PyOpenColorIO.ProcessorCacheFlags attribute*), 298
PROCESSOR_CACHE_SHARE_DYN_PROPERTIES (*PyOpenColorIO.ProcessorCacheFlags attribute*), 298
ProcessorCacheFlags (*class in PyOpenColorIO*), 298
ProcessorMetadata (*class in PyOpenColorIO*), 356

PyOpenColorIO.METADATA_DESCRIPTION (*built-in variable*), 255
PyOpenColorIO.METADATA_ID (*built-in variable*), 256
PyOpenColorIO.METADATA_INFO (*built-in variable*), 255
PyOpenColorIO.METADATA_INPUT_DESCRIPTOR (*built-in variable*), 255
PyOpenColorIO.METADATA_NAME (*built-in variable*), 256
PyOpenColorIO.METADATA_OUTPUT_DESCRIPTOR (*built-in variable*), 255
PyOpenColorIO.OCIO_ACTIVE_DISPLAYS_ENVVAR (*built-in variable*), 253
PyOpenColorIO.OCIO_ACTIVE_VIEWS_ENVVAR (*built-in variable*), 253
PyOpenColorIO.OCIO_CONFIG_ENVVAR (*built-in variable*), 253
PyOpenColorIO.OCIO_DISABLE_ALL_CACHES (*built-in variable*), 256
PyOpenColorIO.OCIO_DISABLE_CACHE_FALLBACK (*built-in variable*), 256
PyOpenColorIO.OCIO_DISABLE_PROCESSOR_CACHES (*built-in variable*), 256
PyOpenColorIO.OCIO_INACTIVE_COLORSPACES_ENVVAR (*built-in variable*), 253
PyOpenColorIO.OCIO_OPTIMIZATION_FLAGS_ENVVAR (*built-in variable*), 253
PyOpenColorIO.OCIO_VIEW_USE_DISPLAY_NAME (*built-in variable*), 255
PyOpenColorIO.ROLE_COLOR_PICKING (*built-in variable*), 254
PyOpenColorIO.ROLE_COLOR_TIMING (*built-in variable*), 254
PyOpenColorIO.ROLE_COMPOSITING_LOG (*built-in variable*), 254
PyOpenColorIO.ROLE_DATA (*built-in variable*), 254
PyOpenColorIO.ROLE_DEFAULT (*built-in variable*), 254
PyOpenColorIO.ROLE_INTERCHANGE_DISPLAY (*built-in variable*), 255
PyOpenColorIO.ROLE_INTERCHANGE_SCENE (*built-in variable*), 254
PyOpenColorIO.ROLE_MATTE_PAINT (*built-in variable*), 254
PyOpenColorIO.ROLE_REFERENCE (*built-in variable*), 254
PyOpenColorIO.ROLE_RENDERING (*built-in variable*), 254
PyOpenColorIO.ROLE_SCENE_LINEAR (*built-in variable*), 254
PyOpenColorIO.ROLE_TEXTURE_PAINT (*built-in variable*), 254

R

RANGE_CLAMP (*PyOpenColorIO.RangeStyle attribute*), 283
RANGE_NO_CLAMP (*PyOpenColorIO.RangeStyle attribute*), 283
RangeStyle (*class in PyOpenColorIO*), 283
RangeStyleFromString() (*in module PyOpenColorIO*), 310
RangeStyleToString() (*in module PyOpenColorIO*), 310
RangeTransform (*class in PyOpenColorIO*), 433
red (*PyOpenColorIO.GradingRGBCurve property*), 319
red (*PyOpenColorIO.GradingRGBM property*), 315
red (*PyOpenColorIO.GradingRGBMSW property*), 333
REFERENCE_SPACE_DISPLAY (*PyOpenColorIO.ReferenceSpaceType attribute*), 268
REFERENCE_SPACE_SCENE (*PyOpenColorIO.ReferenceSpaceType attribute*), 268
ReferenceSpaceType (*class in PyOpenColorIO*), 268
refresh() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
removeActiveDisplay() (*PyOpenColorIO.Config method*), 226
removeActiveView() (*PyOpenColorIO.Config method*), 226
removeAlias() (*PyOpenColorIO.ColorSpace method*), 201
removeAlias() (*PyOpenColorIO.NamedTransform method*), 344
removeCategory() (*PyOpenColorIO.ColorSpace method*), 201
removeCategory() (*PyOpenColorIO.NamedTransform method*), 344
removeCategory() (*PyOpenColorIO.ViewTransform method*), 438
removeColorSpace() (*PyOpenColorIO.ColorSpaceSet method*), 207
removeColorSpace() (*PyOpenColorIO.Config method*), 226
removeColorSpace() (*PyOpenColorIO.ViewingRules method*), 363
removeColorSpaces() (*PyOpenColorIO.ColorSpaceSet method*), 208
RemoveDisplayView() (*in module PyOpenColorIO.DisplayViewHelpers*), 186
removeDisplayView() (*PyOpenColorIO.Config method*), 226
removeEncoding() (*PyOpenColorIO.ViewingRules method*), 363
removeRule() (*PyOpenColorIO.FileRules method*), 359
removeRule() (*PyOpenColorIO.ViewingRules method*), 363
removeSharedView() (*PyOpenColorIO.Config method*), 226
removeVirtualDisplayView() (*PyOpenCol-*

orIO.Config method), 226
ResetComputeHashFunction() (*in module PyOpenColorIO*), 305
ResetToDefaultLoggingFunction() (*in module PyOpenColorIO*), 305
resolveFileLocation() (*PyOpenColorIO.Context method*), 257
resolveStringVar() (*PyOpenColorIO.Context method*), 258
RGB_BLUE (*PyOpenColorIO.RGBCurveType attribute*), 292
RGB_GREEN (*PyOpenColorIO.RGBCurveType attribute*), 292
RGB_MASTER (*PyOpenColorIO.RGBCurveType attribute*), 292
RGB_NUM_CURVES (*PyOpenColorIO.RGBCurveType attribute*), 292
RGB_RED (*PyOpenColorIO.RGBCurveType attribute*), 292
RGBCurveType (*class in PyOpenColorIO*), 292
RoleColorSpaceIterator (*class in PyOpenColorIO.Config*), 230
RoleNameIterator (*class in PyOpenColorIO.Config*), 230

S

samplerName (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
samplerName (*PyOpenColorIO.GpuShaderDesc.Texture3D property*), 381
Sat() (*PyOpenColorIO.MatrixTransform static method*), 430
sat_lum (*PyOpenColorIO.GradingHueCurve property*), 325
sat_sat (*PyOpenColorIO.GradingHueCurve property*), 325
saturation (*PyOpenColorIO.GradingPrimary property*), 314
Scale() (*PyOpenColorIO.MatrixTransform static method*), 430
scontrast (*PyOpenColorIO.GradingTone property*), 332
SEARCH_REFERENCE_SPACE_ALL (*PyOpenColorIO.SearchReferenceSpaceType attribute*), 269
SEARCH_REFERENCE_SPACE_DISPLAY (*PyOpenColorIO.SearchReferenceSpaceType attribute*), 269
SEARCH_REFERENCE_SPACE_SCENE (*PyOpenColorIO.SearchReferenceSpaceType attribute*), 269
SearchPathIterator (*class in PyOpenColorIO.Config*), 229
SearchPathIterator (*class in PyOpenColorIO.Context*), 259
SearchReferenceSpaceType (*class in PyOpenColorIO*), 269
serialize() (*PyOpenColorIO.Config method*), 226
setActiveDisplays() (*PyOpenColorIO.Config method*), 227
setActiveViews() (*PyOpenColorIO.Config method*), 227
setAllocation() (*PyOpenColorIO.AllocationTransform method*), 390
setAllocation() (*PyOpenColorIO.ColorSpace method*), 201
setAllocationVars() (*PyOpenColorIO.ColorSpace method*), 201
setAllowTexture1D() (*PyOpenColorIO.GpuShaderCreator method*), 369
setAllowTexture1D() (*PyOpenColorIO.GpuShaderDesc method*), 380
setAppCategories() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setBase() (*PyOpenColorIO.LogAffineTransform method*), 416
setBase() (*PyOpenColorIO.LogCameraTransform method*), 418
setBase() (*PyOpenColorIO.LogTransform method*), 421
setBitDepth() (*PyOpenColorIO.ColorSpace method*), 201
setBypassLinToLog() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
setCCCId() (*PyOpenColorIO.FileTransform method*), 409
setCDLStyle() (*PyOpenColorIO.FileTransform method*), 409
setChannelView() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
setColorSpace() (*PyOpenColorIO.FileRules method*), 359
setColorTimingCC() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
SetComputeHashFunction() (*in module PyOpenColorIO*), 305
setConfig() (*PyOpenColorIO.Baker method*), 195
setConfig() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setConfigIOProxy() (*PyOpenColorIO.Config method*), 227
setContrast() (*PyOpenColorIO.ExposureContrastTransform method*), 406

setCubeSize() (*PyOpenColorIO.Baker method*), 195
setCustomKey() (*PyOpenColorIO.FileRules method*), 359
setCustomKey() (*PyOpenColorIO.ViewingRules method*), 364
setData() (*PyOpenColorIO.Lut1DTransform method*), 425
setData() (*PyOpenColorIO.Lut3DTransform method*), 428
setDataBypass() (*PyOpenColorIO.ColorSpaceTransform method*), 397
setDataBypass() (*PyOpenColorIO.DisplayViewTransform method*), 399
setDefaultLumaCoefs() (*PyOpenColorIO.Config method*), 227
setDefaultRuleColorSpace() (*PyOpenColorIO.FileRules method*), 359
setDefaultViewTransformName() (*PyOpenColorIO.Config method*), 227
setDescription() (*PyOpenColorIO.ColorSpace method*), 201
setDescription() (*PyOpenColorIO.Config method*), 227
setDescription() (*PyOpenColorIO.Look method*), 342
setDescription() (*PyOpenColorIO.NamedTransform method*), 344
setDescription() (*PyOpenColorIO.ViewTransform method*), 438
setDescriptorSetIndex() (*PyOpenColorIO.GpuShaderCreator method*), 369
setDescriptorSetIndex() (*PyOpenColorIO.GpuShaderDesc method*), 380
setDirection() (*PyOpenColorIO.AllocationTransform method*), 390
setDirection() (*PyOpenColorIO.BuiltinTransform method*), 391
setDirection() (*PyOpenColorIO.CDLTransform method*), 395
setDirection() (*PyOpenColorIO.ColorSpaceTransform method*), 397
setDirection() (*PyOpenColorIO.DisplayViewTransform method*), 399
setDirection() (*PyOpenColorIO.ExponentTransform method*), 401
setDirection() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
setDirection() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setDirection() (*PyOpenColorIO.FileTransform method*), 409
setDirection() (*PyOpenColorIO.FixedFunctionTransform method*), 411
setDirection() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
setDirection() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
setDirection() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
setDirection() (*PyOpenColorIO.GradingToneTransform method*), 330
setDirection() (*PyOpenColorIO.GroupTransform method*), 413
setDirection() (*PyOpenColorIO.LogAffineTransform method*), 416
setDirection() (*PyOpenColorIO.LogCameraTransform method*), 418
setDirection() (*PyOpenColorIO.LogTransform method*), 421
setDirection() (*PyOpenColorIO.LookTransform method*), 422
setDirection() (*PyOpenColorIO.Lut1DTransform method*), 425
setDirection() (*PyOpenColorIO.Lut3DTransform method*), 428
setDirection() (*PyOpenColorIO.MatrixTransform method*), 431
setDirection() (*PyOpenColorIO.RangeTransform method*), 434
setDirection() (*PyOpenColorIO.Transform method*), 388
setDisplay() (*PyOpenColorIO.DisplayViewTransform method*), 399
setDisplayCC() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
setDisplayTemporary() (*PyOpenColorIO.Config method*), 227
setDisplayView() (*PyOpenColorIO.Baker method*), 195
setDisplayViewTransform() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
setDouble() (*PyOpenColorIO.DynamicProperty method*), 263
setDrawCurveOnly() (*PyOpenColorIO.GradingHueCurve method*), 325
setDst() (*PyOpenColorIO.ColorSpaceTransform method*), 397
setDst() (*PyOpenColorIO.LookTransform method*), 422
setElementName() (*PyOpenColorIO.FormatMetadata method*), 301
setElementValue() (*PyOpenColorIO.FormatMetadata method*), 301

setEncoding() (*PyOpenColorIO.ColorSpace method*), 201
setEncoding() (*PyOpenColorIO.NamedTransform method*), 344
setEncodings() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setEnvironmentMode() (*PyOpenColorIO.Config method*), 227
setEnvironmentMode() (*PyOpenColorIO.Context method*), 258
SetEnvVariable() (*in module PyOpenColorIO*), 306
setEqualityGroup() (*PyOpenColorIO.ColorSpace method*), 201
setExposure() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setExtension() (*PyOpenColorIO.FileRules method*), 359
setFamily() (*PyOpenColorIO.ColorSpace method*), 201
setFamily() (*PyOpenColorIO.NamedTransform method*), 344
setFamily() (*PyOpenColorIO.ViewTransform method*), 438
setFamilySeparator() (*PyOpenColorIO.Config method*), 228
setFileInputBitDepth() (*PyOpenColorIO.MatrixTransform method*), 431
setFileInputBitDepth() (*PyOpenColorIO.RangeTransform method*), 434
setFileOutputBitDepth() (*PyOpenColorIO.Lut1DTransform method*), 425
setFileOutputBitDepth() (*PyOpenColorIO.Lut3DTransform method*), 428
setFileOutputBitDepth() (*PyOpenColorIO.MatrixTransform method*), 431
setFileOutputBitDepth() (*PyOpenColorIO.RangeTransform method*), 434
setFileRules() (*PyOpenColorIO.Config method*), 228
setFirstSOPDescription() (*PyOpenColorIO.CDLTransform method*), 395
setFormat() (*PyOpenColorIO.Baker method*), 195
setFunctionName() (*PyOpenColorIO.GpuShaderCreator method*), 369
setFunctionName() (*PyOpenColorIO.GpuShaderDesc method*), 380
setGamma() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
setGamma() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setGradingHueCurve() (*PyOpenColorIO.DynamicProperty method*), 263
setGradingPrimary() (*PyOpenColorIO.DynamicProperty method*), 263
setGradingRGBCurve() (*PyOpenColorIO.DynamicProperty method*), 263
setGradingTone() (*PyOpenColorIO.DynamicProperty method*), 263
setGridSize() (*PyOpenColorIO.Lut3DTransform method*), 428
setHueAdjust() (*PyOpenColorIO.Lut1DTransform method*), 425
setID() (*PyOpenColorIO.CDLTransform method*), 395
setID() (*PyOpenColorIO.FormatMetadata method*), 301
setInactiveColorSpaces() (*PyOpenColorIO.Config method*), 228
setIncludeColorSpaces() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setIncludeNamedTransforms() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setIncludeRoles() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setInputHalfDomain() (*PyOpenColorIO.Lut1DTransform method*), 425
setInputSpace() (*PyOpenColorIO.Baker method*), 195
setInterchangeAttribute() (*PyOpenColorIO.ColorSpace method*), 201
setInterchangeAttribute() (*PyOpenColorIO.Look method*), 342
setInterchangeAttribute() (*PyOpenColorIO.ViewTransform method*), 438
setInteropID() (*PyOpenColorIO.ColorSpace method*), 201
setInterpolation() (*PyOpenColorIO.FileTransform method*), 409
setInterpolation() (*PyOpenColorIO.Lut1DTransform method*), 425
setInterpolation() (*PyOpenColorIO.Lut3DTransform method*), 428
setInverseTransform() (*PyOpenColorIO.Look method*), 342
setIsData() (*PyOpenColorIO.ColorSpace method*), 201
setLanguage() (*PyOpenColorIO.GpuShaderCreator method*), 369
setLanguage() (*PyOpenColorIO.GpuShaderDesc method*), 380
setLength() (*PyOpenColorIO.Lut1DTransform method*), 425
setLinearCC() (*PyOpenCol-*

orIO.LegacyViewingPipeline method), 188
setLinearSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 418
setLinSideBreakValue() (*PyOpenColorIO.LogCameraTransform method*), 418
setLinSideOffsetValue() (*PyOpenColorIO.LogAffineTransform method*), 416
setLinSideOffsetValue() (*PyOpenColorIO.LogCameraTransform method*), 418
setLinSideSlopeValue() (*PyOpenColorIO.LogAffineTransform method*), 416
setLinSideSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 418
setLogExposureStep() (*PyOpenColorIO.ExposureContrastTransform method*), 406
SetLoggingFunction() (*in module PyOpenColorIO*), 305
SetLoggingLevel() (*in module PyOpenColorIO*), 304
setLogMidGray() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setLogSideOffsetValue() (*PyOpenColorIO.LogAffineTransform method*), 416
setLogSideOffsetValue() (*PyOpenColorIO.LogCameraTransform method*), 419
setLogSideSlopeValue() (*PyOpenColorIO.LogAffineTransform method*), 416
setLogSideSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 419
setLooks() (*PyOpenColorIO.Baker method*), 195
setLooks() (*PyOpenColorIO.LookTransform method*), 422
setLooksBypass() (*PyOpenColorIO.DisplayViewTransform method*), 399
setLooksOverride() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
setLooksOverrideEnabled() (*PyOpenColorIO.LegacyViewingPipeline method*), 188
setMajorVersion() (*PyOpenColorIO.Config method*), 228
setMatrix() (*PyOpenColorIO.MatrixTransform method*), 431
setMaxInValue() (*PyOpenColorIO.RangeTransform method*), 434
setMaxOutValue() (*PyOpenColorIO.RangeTransform method*), 434
setMinInValue() (*PyOpenColorIO.RangeTransform method*), 434
setMinorVersion() (*PyOpenColorIO.Config method*), 228
setMinOutValue() (*PyOpenColorIO.RangeTransform method*), 434
setName() (*PyOpenColorIO.ColorSpace method*), 201
setName() (*PyOpenColorIO.Config method*), 228
setName() (*PyOpenColorIO.FormatMetadata method*), 301
setName() (*PyOpenColorIO.Look method*), 342
setName() (*PyOpenColorIO.NamedTransform method*), 344
setName() (*PyOpenColorIO.ViewTransform method*), 438
setNegativeStyle() (*PyOpenColorIO.ExponentTransform method*), 401
setNegativeStyle() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
setNumControlPoints() (*PyOpenColorIO.GradingBSplineCurve method*), 328
setOffset() (*PyOpenColorIO.CDLTransform method*), 395
setOffset() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
setOffset() (*PyOpenColorIO.MatrixTransform method*), 431
setOutputRawHalfs() (*PyOpenColorIO.Lut1DTransform method*), 425
setParams() (*PyOpenColorIO.FixedFunctionTransform method*), 411
setPattern() (*PyOpenColorIO.FileRules method*), 359
setPivot() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setPixelName() (*PyOpenColorIO.GpuShaderCreator method*), 369
setPixelName() (*PyOpenColorIO.GpuShaderDesc method*), 380
setPower() (*PyOpenColorIO.CDLTransform method*), 395
setProcessorCacheFlags() (*PyOpenColorIO.Config method*), 228
setProcessSpace() (*PyOpenColorIO.Look method*), 342
setRegex() (*PyOpenColorIO.FileRules method*), 359
setResourcePrefix() (*PyOpenColorIO.GpuShaderCreator method*), 369
setResourcePrefix() (*PyOpenColorIO.GpuShaderDesc method*), 380
setRGBToHSY() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
setRole() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setRole() (*PyOpenColorIO.Config method*), 228
setSat() (*PyOpenColorIO.CDLTransform method*), 395

setSearchPath() (*PyOpenColorIO.Config method*), 228
setSearchPath() (*PyOpenColorIO.Context method*), 258
setSearchReferenceSpaceType() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setSelectedMixingEncoding() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
setSelectedMixingEncodingIdx() (*PyOpenColorIO.MixingColorSpaceManager method*), 191
setSelectedMixingSpace() (*PyOpenColorIO.MixingColorSpaceManager method*), 192
setSelectedMixingSpaceIdx() (*PyOpenColorIO.MixingColorSpaceManager method*), 192
setShaperSize() (*PyOpenColorIO.Baker method*), 195
setShaperSpace() (*PyOpenColorIO.Baker method*), 195
setSkipColorSpaceConversion() (*PyOpenColorIO.LookTransform method*), 422
setSliderMaxEdge() (*PyOpenColorIO.MixingSlider method*), 191
setSliderMinEdge() (*PyOpenColorIO.MixingSlider method*), 191
setSlope() (*PyOpenColorIO.CDLTransform method*), 395
setSlope() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
setSlope() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
setSlopes() (*PyOpenColorIO.GradingBSplineCurve method*), 328
setSOP() (*PyOpenColorIO.CDLTransform method*), 395
setSplineType() (*PyOpenColorIO.GradingBSplineCurve method*), 328
setSrc() (*PyOpenColorIO.ColorSpaceTransform method*), 397
setSrc() (*PyOpenColorIO.DisplayViewTransform method*), 399
setSrc() (*PyOpenColorIO.FileTransform method*), 409
setSrc() (*PyOpenColorIO.LookTransform method*), 422
setStrictParsingEnabled() (*PyOpenColorIO.Config method*), 228
setStringVar() (*PyOpenColorIO.Context method*), 258
setStyle() (*PyOpenColorIO.BuiltinTransform method*), 391
setStyle() (*PyOpenColorIO.CDLTransform method*), 395
setStyle() (*PyOpenColorIO.ExposureContrastTransform method*), 406
setStyle() (*PyOpenColorIO.FixedFunctionTransform method*), 411
setStyle() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
setStyle() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
setStyle() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
setStyle() (*PyOpenColorIO.GradingToneTransform method*), 330
setStyle() (*PyOpenColorIO.RangeTransform method*), 435
setTargetSpace() (*PyOpenColorIO.Baker method*), 196
setTextureMaxWidth() (*PyOpenColorIO.GpuShaderCreator method*), 369
setTextureMaxWidth() (*PyOpenColorIO.GpuShaderDesc method*), 380
setTransform() (*PyOpenColorIO.ColorSpace method*), 201
setTransform() (*PyOpenColorIO.Look method*), 342
setTransform() (*PyOpenColorIO.NamedTransform method*), 344
setTransform() (*PyOpenColorIO.ViewTransform method*), 438
setTreatNoCategoryAsAny() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setUniqueID() (*PyOpenColorIO.GpuShaderCreator method*), 369
setUniqueID() (*PyOpenColorIO.GpuShaderDesc method*), 380
setUserCategories() (*PyOpenColorIO.ColorSpaceMenuParameters method*), 180
setValue() (*PyOpenColorIO.ExponentTransform method*), 401
setValue() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
setValue() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
setValue() (*PyOpenCol-*

orIO.GradingRGBCurveTransform method), 317
setValue() (*PyOpenColorIO.GradingToneTransform method*), 330
setValue() (*PyOpenColorIO.Lut1DTransform method*), 425
setValue() (*PyOpenColorIO.Lut3DTransform method*), 428
setVars() (*PyOpenColorIO.AllocationTransform method*), 390
setVersion() (*PyOpenColorIO.Config method*), 228
setView() (*PyOpenColorIO.DisplayViewTransform method*), 399
setViewingRules() (*PyOpenColorIO.Config method*), 228
setWorkingDir() (*PyOpenColorIO.Config method*), 229
setWorkingDir() (*PyOpenColorIO.Context method*), 258
shadows (*PyOpenColorIO.GradingTone property*), 332
SharedViewIterator (*class in PyOpenColorIO.Config*), 230
sliderToMixing() (*PyOpenColorIO.MixingSlider method*), 191
slopesAreDefault() (*PyOpenColorIO.GradingBSplineCurve method*), 328
slopesAreDefault() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
slopesAreDefault() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
start (*PyOpenColorIO.GradingRGBMSW property*), 333
StringVarIterator (*class in PyOpenColorIO.Context*), 258
StringVarNameIterator (*class in PyOpenColorIO.Context*), 258
SystemMonitors (*class in PyOpenColorIO*), 387

T

Texture (*class in PyOpenColorIO.GpuShaderDesc*), 381
Texture3D (*class in PyOpenColorIO.GpuShaderDesc*), 381
Texture3DIterator (*class in PyOpenColorIO.GpuShaderDesc*), 382
TEXTURE_1D (*PyOpenColorIO.GpuShaderCreator attribute*), 367
TEXTURE_1D (*PyOpenColorIO.GpuShaderCreator.TextureDimensions attribute*), 367
TEXTURE_1D (*PyOpenColorIO.GpuShaderDesc attribute*), 377
TEXTURE_1D (*PyOpenColorIO.GpuShaderDesc.TextureDimensions attribute*), 377
TEXTURE_2D (*PyOpenColorIO.GpuShaderCreator attribute*), 367
TEXTURE_2D (*PyOpenColorIO.GpuShaderCreator.TextureDimensions attribute*), 367
TEXTURE_2D (*PyOpenColorIO.GpuShaderDesc attribute*), 377
TEXTURE_2D (*PyOpenColorIO.GpuShaderDesc.TextureDimensions attribute*), 377
TEXTURE_RED_CHANNEL (*PyOpenColorIO.GpuShaderCreator.TextureType attribute*), 369
TEXTURE_RED_CHANNEL (*PyOpenColorIO.GpuShaderDesc.TextureType attribute*), 380
TEXTURE_RGB_CHANNEL (*PyOpenColorIO.GpuShaderCreator.TextureType attribute*), 369
TEXTURE_RGB_CHANNEL (*PyOpenColorIO.GpuShaderDesc.TextureType attribute*), 380
TextureIterator (*class in PyOpenColorIO.GpuShaderDesc*), 381
textureName (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
textureName (*PyOpenColorIO.GpuShaderDesc.Texture3D property*), 381
textureShaderBindingIndex (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
textureShaderBindingIndex (*PyOpenColorIO.GpuShaderDesc.Texture3D property*), 381
TextureType (*class in PyOpenColorIO.GpuShaderCreator*), 369
TextureType (*class in PyOpenColorIO.GpuShaderDesc*), 380
Transform (*class in PyOpenColorIO*), 388
TRANSFORM_DIR_FORWARD (*PyOpenColorIO.TransformDirection attribute*), 272
TRANSFORM_DIR_INVERSE (*PyOpenColorIO.TransformDirection attribute*), 272
TRANSFORM_TYPE_ALLOCATION (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_BUILTIN (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_CDL (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_COLORSPACE (*PyOpenCol-*

orIO.TransformType attribute), 273
TRANSFORM_TYPE_DISPLAY_VIEW (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_EXPONENT (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_EXPONENT_WITH_LINEAR (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_EXPOSURE_CONTRAST (*PyOpenColorIO.TransformType attribute*), 273
TRANSFORM_TYPE_FILE (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_FIXED_FUNCTION (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_GRADING_HUE_CURVE (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_GRADING_PRIMARY (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_GRADING_RGB_CURVE (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_GRADING_TONE (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_GROUP (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LOG (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LOG_AFFINE (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LOG_CAMERA (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LOOK (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LUT1D (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_LUT3D (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_MATRIX (*PyOpenColorIO.TransformType attribute*), 274
TRANSFORM_TYPE_RANGE (*PyOpenColorIO.TransformType attribute*), 274
TransformDirection (*class in PyOpenColorIO*), 272
TransformDirectionFromString() (*in module PyOpenColorIO*), 307
TransformDirectionToString() (*in module PyOpenColorIO*), 307
TransformFormatMetadataIterator (*class in PyOpenColorIO.Processor*), 349
TransformIterator (*class in PyOpenColorIO.GroupTransform*), 414
TransformType (*class in PyOpenColorIO*), 273
type (*PyOpenColorIO.GpuShaderDesc.UniformData property*), 381

U

UNIFORM_BOOL (*PyOpenColorIO.UniformDataType attribute*), 293
UNIFORM_DOUBLE (*PyOpenColorIO.UniformDataType attribute*), 293
UNIFORM_FLOAT3 (*PyOpenColorIO.UniformDataType attribute*), 293
UNIFORM_UNKNOWN (*PyOpenColorIO.UniformDataType attribute*), 293
UNIFORM_VECTOR_FLOAT (*PyOpenColorIO.UniformDataType attribute*), 293
UNIFORM_VECTOR_INT (*PyOpenColorIO.UniformDataType attribute*), 293
UniformData (*class in PyOpenColorIO.GpuShaderDesc*), 380
UniformDataType (*class in PyOpenColorIO*), 293
UniformIterator (*class in PyOpenColorIO.GpuShaderDesc*), 381
UnsetEnvVariable() (*in module PyOpenColorIO*), 306
unsetLinearSlopeValue() (*PyOpenColorIO.LogCameraTransform method*), 419
unsetMaxInValue() (*PyOpenColorIO.RangeTransform method*), 435
unsetMaxOutValue() (*PyOpenColorIO.RangeTransform method*), 435
unsetMinInValue() (*PyOpenColorIO.RangeTransform method*), 435
unsetMinOutValue() (*PyOpenColorIO.RangeTransform method*), 435
upgradeToLatestVersion() (*PyOpenColorIO.Config method*), 229

V

validate() (*PyOpenColorIO.AllocationTransform method*), 390
validate() (*PyOpenColorIO.BuiltinTransform method*), 391
validate() (*PyOpenColorIO.CDLTransform method*), 395
validate() (*PyOpenColorIO.ColorSpaceTransform method*), 397
validate() (*PyOpenColorIO.Config method*), 229
validate() (*PyOpenColorIO.DisplayViewTransform method*), 399
validate() (*PyOpenColorIO.ExponentTransform method*), 401
validate() (*PyOpenColorIO.ExponentWithLinearTransform method*), 403
validate() (*PyOpenColorIO.ExposureContrastTransform method*), 406
validate() (*PyOpenColorIO.FileTransform method*), 409
validate() (*PyOpenColorIO.FixedFunctionTransform method*), 411

validate() (*PyOpenColorIO.GradingBSplineCurve method*), 328
validate() (*PyOpenColorIO.GradingHueCurve method*), 325
validate() (*PyOpenColorIO.GradingHueCurveTransform method*), 322
validate() (*PyOpenColorIO.GradingPrimary method*), 314
validate() (*PyOpenColorIO.GradingPrimaryTransform method*), 312
validate() (*PyOpenColorIO.GradingRGBCurve method*), 319
validate() (*PyOpenColorIO.GradingRGBCurveTransform method*), 317
validate() (*PyOpenColorIO.GradingTone method*), 332
validate() (*PyOpenColorIO.GradingToneTransform method*), 330
validate() (*PyOpenColorIO.GroupTransform method*), 413
validate() (*PyOpenColorIO.LogAffineTransform method*), 416
validate() (*PyOpenColorIO.LogCameraTransform method*), 419
validate() (*PyOpenColorIO.LogTransform method*), 421
validate() (*PyOpenColorIO.LookTransform method*), 422
validate() (*PyOpenColorIO.Lut1DTransform method*), 425
validate() (*PyOpenColorIO.Lut3DTransform method*), 428
validate() (*PyOpenColorIO.MatrixTransform method*), 431
validate() (*PyOpenColorIO.RangeTransform method*), 435
validate() (*PyOpenColorIO.Transform method*), 388
value (*PyOpenColorIO.Allocation property*), 280
value (*PyOpenColorIO.BitDepth property*), 278
value (*PyOpenColorIO.CDLStyle property*), 288
value (*PyOpenColorIO.ChannelOrdering property*), 279
value (*PyOpenColorIO.ColorSpaceDirection property*), 271
value (*PyOpenColorIO.ColorSpaceVisibility property*), 270
value (*PyOpenColorIO.DynamicPropertyType property*), 291
value (*PyOpenColorIO.EnvironmentMode property*), 283
value (*PyOpenColorIO.ExposureContrastStyle property*), 288
value (*PyOpenColorIO.FixedFunctionStyle property*), 286
value (*PyOpenColorIO.GpuLanguage property*), 281
value (*PyOpenColorIO.GpuShaderCreator.TextureDimensions property*), 367
value (*PyOpenColorIO.GpuShaderCreator.TextureType property*), 369
value (*PyOpenColorIO.GpuShaderDesc.TextureDimensions property*), 377
value (*PyOpenColorIO.GpuShaderDesc.TextureType property*), 380
value (*PyOpenColorIO.GradingStyle property*), 290
value (*PyOpenColorIO.Interpolation property*), 276
value (*PyOpenColorIO.LoggingLevel property*), 267
value (*PyOpenColorIO.Lut1DHueAdjust property*), 279
value (*PyOpenColorIO.NamedTransformVisibility property*), 270
value (*PyOpenColorIO.NegativeStyle property*), 289
value (*PyOpenColorIO.OptimizationFlags property*), 296
value (*PyOpenColorIO.ProcessorCacheFlags property*), 298
value (*PyOpenColorIO.RangeStyle property*), 283
value (*PyOpenColorIO.ReferenceSpaceType property*), 268
value (*PyOpenColorIO.RGBCurveType property*), 292
value (*PyOpenColorIO.SearchReferenceSpaceType property*), 269
value (*PyOpenColorIO.TransformDirection property*), 272
value (*PyOpenColorIO.TransformType property*), 274
value (*PyOpenColorIO.UniformDataType property*), 293
value (*PyOpenColorIO.ViewTransformDirection property*), 272
value (*PyOpenColorIO.ViewType property*), 271
View() (*PyOpenColorIO.MatrixTransform static method*), 430
VIEW_DISPLAY_DEFINED (*PyOpenColorIO.ViewType attribute*), 271
VIEW_SHARED (*PyOpenColorIO.ViewType attribute*), 271
ViewForColorSpaceIterator (*class in PyOpenColorIO.Config*), 231
ViewingRuleColorSpaceIterator (*class in PyOpenColorIO.ViewingRules*), 364
ViewingRuleEncodingIterator (*class in PyOpenColorIO.ViewingRules*), 364
ViewingRules (*class in PyOpenColorIO*), 362
ViewIterator (*class in PyOpenColorIO.Config*), 230
ViewTransform (*class in PyOpenColorIO*), 437
VIEWTRANSFORM_DIR_FROM_REFERENCE (*PyOpenColorIO.ViewTransformDirection attribute*), 272
VIEWTRANSFORM_DIR_TO_REFERENCE (*PyOpenCol-*

orIO.ViewTransformDirection attribute), 272
ViewTransformCategoryIterator (*class in PyOpenColorIO.ViewTransform*), 438
ViewTransformDirection (*class in PyOpenColorIO*), 272
ViewTransformIterator (*class in PyOpenColorIO.Config*), 231
ViewTransformNameIterator (*class in PyOpenColorIO.Config*), 231
ViewType (*class in PyOpenColorIO*), 271

W

whites (*PyOpenColorIO.GradingTone property*), 332
width (*PyOpenColorIO.GpuShaderDesc.Texture property*), 381
width (*PyOpenColorIO.GradingRGBMSW property*), 333
write() (*PyOpenColorIO.GroupTransform method*), 413
WriteFormatIterator (*class in PyOpenColorIO.GroupTransform*), 414

X

x (*PyOpenColorIO.GradingControlPoint property*), 327

Y

y (*PyOpenColorIO.GradingControlPoint property*), 327