
Implementation Notes for the IccProfLib Color Management Module (CMM) in iccDEV

The iccDEV project is an open source object oriented C++ development project that is available at <https://github.com/InternationalColorConsortium/iccDEV>. It was written to provide an example of how various aspects of color management can be implemented. The basis of iccDEV was originally written by Max Derhak as a class project for the Color Systems course while pursuing an MS degree at the Rochester Institute of Technology. After extensive revisions (directed by the ICC Architecture Working Group) the project was turned over to the International Color Consortium as a means of helping to describe color management implementation approaches inferred by the ICC Color profile specification (see <http://www.color.org>).

iccDEV contains a platform independent library (named IccProfLib) that provides a complete implementation for reading, writing, and applying ICC profiles. The IccProfLib sub-project has HTML documentation that describes the classes and their interfaces, but the basic relationship between the classes as it relates to applying profiles is not necessarily clear. This document complements the IccProfLib class documentation by describing how the objects interact when applying profiles. This document assumes familiarity with both object oriented programming and the ICC profile specification. Overview information will be given related to classes within IccProfLib. For specific details consult the implementation as defined by the source code. (Note: IccProfLib was initially named IccLib, but has since been changed to avoid conflicts with existing Libraries).

There are multiple ways to go about implementing Color Management (See ICC White Paper #7 – The role of ICC profiles in a colour reproduction system). The implementation presented in this document represents the fulfillment of a ‘Dumb’ CMM with the smarts of color rendering contained in the ICC profiles themselves. This does not preclude the possibility of implementing a ‘Smart’ CMM based upon the profile file support provided by IccProfLib.

Overview

The following discussion makes use of Figure 1 (on following page) to help explain how profiles are applied within lccProfLib.

ICC profile files are read, written, and otherwise manipulated through the use of ClccProfile objects that have attached ClccTag objects which contain the associated profile tag data. The application of profiles is implemented separately through the use of a ClccCmm class object.

A ClccCmm class object is used to administer and perform color management transforms. This object manages a list of ClccXform derived objects which are associated with corresponding ClccProfile objects. Each ClccXform object obtains information from its corresponding ClccProfile object and attached ClccTag objects in order to perform the requested color transformations.

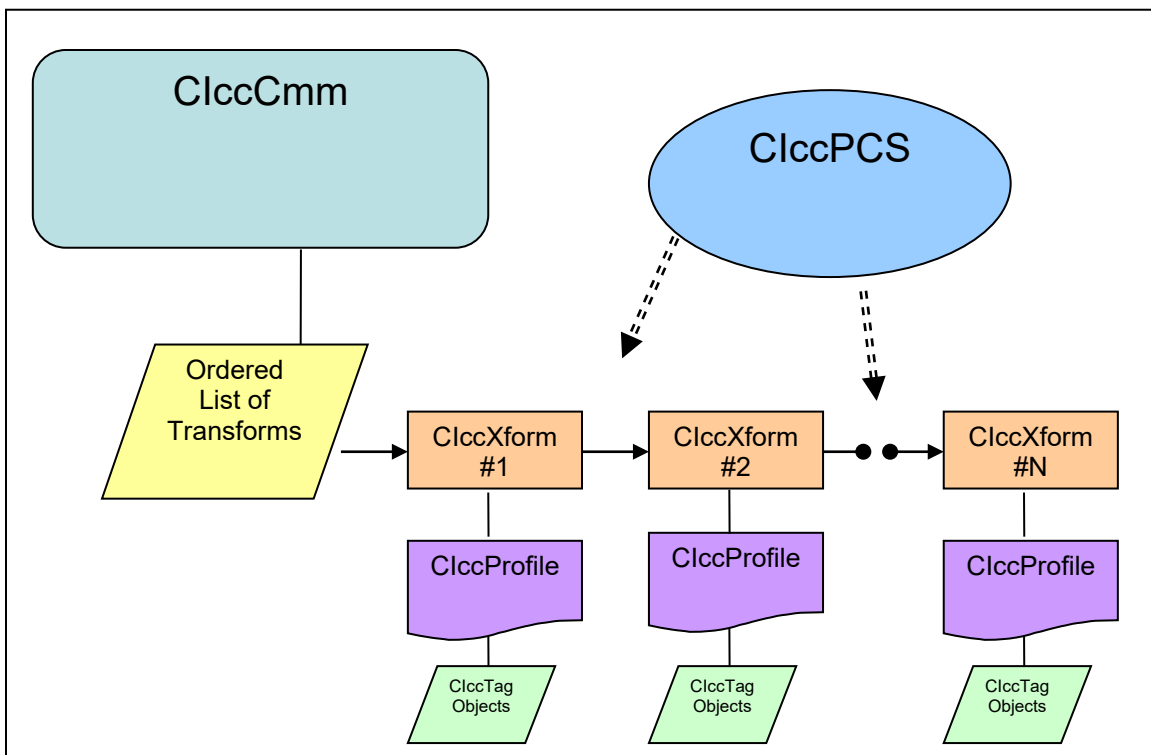


Figure 1 - Object relationships in lccProfLib

Profile application is performed by using the ClccCmm::**Apply()** method after a ClccCmm object has been properly constructed and initialized. This method makes use of the list of ClccXform objects with their associated ClccProfile objects to perform color space transforms.

When applying pixel colors, the ClccCmm object assumes that data pixels have been converted to an internal floating point encoding with all values ranging from 0.0 to 1.0. Results from calling the ClccCmm::**Apply()** method are also in this range. The ClccCmm class provides overloaded conversion member methods (ClccCmm::**ToInternalEncoding()** and ClccCmm::**FromInternalEncoding()**) to

facilitate conversion to/from typical encoding range values used by various pixel formats. The ClccCmm class provides the static Boolean member function ClccCmm::IsInGamut() to determine whether the internal representation of the result is in Gamut when using a Gamut tag from a profile.

ClccCmm Details

There are five stages in the life of a ClccCmm object.

1. Creation – The following information is provided to a ClccCmm object when it is created:
 - a. The source and destination color spaces are identified. In many cases the color spaces can be specified as undefined – the color spaces will be determined by the attached profiles.
 - b. In preparation for the next stage, the initial transform side (Input vs Output) is identified.
2. Attachment – One or more calls to a ClccCmm::AddXform() method are used to attach one or more ICC profiles to the ClccCmm object. There are several overloaded versions of this method:
 - a. In one version the first argument is the file path to the ICC profile file.
 - b. In another overloaded version the first argument is a pointer to ClccProfile object with the profile already loaded (in which case the ownership of the ClccProfile object is passed to the ClccCmm object).
 - c. In another overloaded version passes a reference to a ClccProfile object (in which case the ClccProfile object is copied with the copy owned by the ClccCmm Object).
 - d. In another version the first argument is a pointer to a memory based ICC profile file with the second argument being the length of the file in memory.

Regardless of which version is used, the ClccCmm object keeps track of whether the input or output side of an attached profile should be used. It also keeps track of the connecting color spaces and ensures compatibility. Any number of profiles can be attached to a ClccCmm object. The order in which profiles are attached to the ClccCmm object defines the order that the appropriate transforms will be applied.

Only a single transform from a profile will be used for each attached profile. Since multiple transforms (in separate tags) can be stored in a single ICC profile, ClccCmm::AddXform() arguments are used to determine which transform should be used. The nIntent argument allows selection between rendering intents, and the nLutType argument allows

selection between the Color, Preview, and Gamut tags. NamedColor profile selection is automatically detected when using the basic Color transforms. (Note: The Preview and Gamut transforms are considered to be output transforms for automatic input/output transform selection purposes).

ClccCmm::**AddXform()** creates a ClccXform object for each profile as it is attached to both keep track of the profile and provide the implementation of the transformation using data from the profile.

3. Initialization – The ClccCmm::**Begin()** method is used to indicate that no more profiles will be attached, and that color transformation processing will now begin. The ClccCmm::**Begin()** method performs final color space verification and then each attached ClccXform object is initialized (using the ClccXform::**Begin()** method) to begin color transformations.
4. Apply – The ClccCmm::**Apply()** method applies the ordered sequence of ClccXform objects to the source pixel to arrive at destination pixel values. This method uses a ClccPCS object with the initial color space to keep track of the current color space as transforms are applied. A temporary pixel is also defined and modified within the method. The source pixel, temporary pixel, and destination pixel are all involved in the concept of the “current” pixel. For each transform in the ordered ClccXform object list the “current” pixel is checked with the ClccPCS::**CheckPCS()** method to make sure that the current color space agrees with the input color space of the next transform. The adjusted pixel is then passed to the ClccXform::**Apply()** method to perform the pixel transformation. Once the last transform is performed, the ClccPCS::**CheckLast()** method is used to make any final color space adjustments.
5. Destruction – The ClccXform list and its accompanying objects are released.

Note: IccProfLib provides support for all color profile types (ICC.1:2004-10 Section 8.3 through 8.9). All color profile types except Named color profiles (ICC.1:2004-10 Section 8.9) are supported by the ClccCmm class. The ClccNamedCmm class (also defined in IccCmm.cpp) is derived from ClccCmm class and supports the use of named profiles in addition to the capabilities offered by the ClccCmm class. This was done to avoid the cost of the extra overhead of supporting named colors in the basic CMM class as defined by ClccCmm. The ClccNamedCmm class provides additional ClccNamedCmm::**Apply()** method interfaces to support the input and/or output of color names. This approach allows multiple Named Color profiles to be linked together using color names as a connection space.

ClccXform Details

ClccXform is the base class that defines the basic interface for performing pixel transformations. There are multiple classes that are derived from this class that

provide specific implementations. There are three important static member methods for the ClccXform base class: They include:

1. The static member function ClccXform::**Create()** is used to create actual instances of ClccXform objects. This function uses a ClccProfile object argument to decide which specific ClccXform object to create. The type of ClccXform depends upon the type of transform that is implied by the ICC Profile. Three types are possible: Matrix/TRC, Multi-dimensional lookup table, and Named Color indexing. The ClccXform choices include:
 - ClccXformMatrixTRC – Uses the RGB chromaticities and transfer functions to perform pixel transforms.
 - ClccXform3DLut – Performs pixel transformation on 3D input data. The extracted tag from the attached ClccProfile is determined by the rendering intent and input/output flag. The ClccXform3DLut object is also configured to perform either linear or tetrahedral interpolation.
 - ClccXform4DLut – Performs pixel transformation on 4D input data. The extracted tag from the attached ClccProfile is determined by the rendering intent and input/output flag. The ClccXform4DLut object only performs linear interpolation.
 - ClccXformNDLut – Performs pixel transformation on N-dimensional input data. The extracted tag from the attached ClccProfile is determined by the rendering intent and input/output flag. The ClccXformNDLut object only performs linear interpolation.
 - ClccXformNamedColor – Performs color transforms using text strings to define the color. The static ClccXform::**Create()** method is passed an argument that specifies whether or not the calling ClccCmm object supports named colors. If named colors are not supported, then this object type will not be created.
2. The protected member method ClccXform::**CheckSrcAbs()** is called by the derived ClccXform::**Apply()** methods to perform any required absolute to relative colorimetry transformation. This method also handles legacy PCS encoding, and Version 4 to Version 2 Perceptual black point translation. (Note: If the source color space is not a PCS color space this method makes no adjustments to the pixel).
3. The protected member method ClccXform::**CheckDstAbs()** is called by the derived ClccXform::**Apply()** methods to perform any required relative to absolute colorimetry transformation. This method also handles legacy PCS encoding, and Version 2 to Version 4 Perceptual black point translation. (Note: If the destination color space is not a PCS color space this method makes no adjustments to the pixel).

There are two virtual methods that all derived ClccXform objects need to implement. These are:

1. The virtual `ClccXform::Begin()` method is called during `ClccCmm::Init()` to allow the `ClccXform` derived object to initialize itself relative to the attached color spaces, input/output transform flag, and rendering intent. Additional important methods that are also used include:
 - `ClccXformMatrixTRC` – The `ClccXformMatrixTRC::Begin()` method calculates a matrix and 1D LUT's to use. In some cases an inverse matrix and LUT's are calculated. (See implementation for details).
 - `ClccXform3DLut`, `ClccXform4DLut`, `ClccXformNDLut` – Extracts appropriate curve and LUT tags from the profile and prepares for pixel transformations.
 - `ClccXformNamedColor` – Identifies the correct `ClccXformNamedColor::Apply()` interface to use based upon attached color spaces.

2. A virtual `ClccXform::Apply()` method does most of the work of color transformation. Each derived object provides the implementation of this method to perform the specific operations that are required to implement the color transformation. The order of the operations depends upon whether the `ClccXform` object represents an input transformation or an output transformation. The operations by transform type are as follows:
 - `ClccXformMatrixTRC` – If the `ClccXform` object represents an input transform the following steps are performed:
 - a. `ClccXform::CheckSrcPCS()`
 - b. Apply 1D curves lookup
 - c. Apply matrix
 - d. `ClccXform::CheckDstPCS()`

If the `Xform` represents an output transform, the following steps are performed:

 - a. `ClccXform::CheckSrcPCS()`
 - b. Apply matrix
 - c. Apply 1D curves lookup
 - d. `ClccXform::CheckDstPCS()`

 - `ClccXform3DLut`, `ClccXform4DLut`, `ClccXformNDLut` – The following lists show the order of operations. Not all profile tags provide data to perform operations in which case steps associated with missing data are simply ignored.

If the `ClccXform` object represents an input transform the following steps are performed:

- a. ClccXform::**CheckSrcPCS()**
- b. Apply 1D B curves lookup
- c. Apply matrix
- d. Apply 1D M curves lookup
- e. Perform multi-dimensional interpolation
- f. Apply 1D A curves lookup
- g. ClccXform::**CheckDstPCS()**

If the Xform represents an output transform, the following steps are performed:

- a. ClccXform::**CheckSrcPCS()**
- b. Apply 1D A curves lookup
- c. Perform multi-dimensional interpolation
- d. Apply 1D M curves lookup
- e. Apply matrix
- f. Apply 1D B curves lookup
- g. ClccXform::**CheckDstPCS()**

- ClccXformNamedColor –This object type uses the ClccTagNamedColor2 tag object of the associated ClccProfile to perform the color transformations. The ClccXformNamedColor object behaves differently than the other ClccXform object types. Different ClccXformNamedColor::**Apply()** interfaces are supported to allow for transforms involving named colors. It requires that a named color is always used as either the input or the output side of the transform. Thus direct transforms to/from device coordinates from/to PCS values are not directly supported. (Note: To accomplish this simply attach the named profile to a ClccCmm object twice - the first time with the named color as the output, and the second time with the named color as the input. This results in two ClccXformNamedColor objects being used).

If the input color space is a named color space the operations are as follows:

- a. Search for color name in the Named Color tag.
- b. If the output color space is PCS then set pixel to corresponding PCS value and apply ClccXform::**CheckDstPCS()**.
- c. Else (the output color space is a device color space) set pixel to corresponding device values.

If the output color space is a named color space the operations are as follows:

- a. If the input color space is PCS then call `ClccXform::CheckSrcPCS()` and then find the color index of the color whose PCS value has the least ΔE difference to the source color.
- b. Else (the input color space is a device color space) find the color index of the color whose device coordinate has the smallest Euclidean distance to the source color.
- c. Set destination color to the corresponding index color name.

ClccPCS Details

The `ClccPCS` object is a disposable object that is used to keep track of the current color space as transformations are applied within the `ClccCmm::Apply()` method. In addition to storing the current color space this object also performs necessary PCS conversions when connecting profiles with different PCS characteristics. Such differences include CIE XYZ, CIE Lab Legacy, and CIE Lab Version 4 encodings. Since each of these “color spaces” is considered to be a Profile Connection Space the `ClccPCS` object is used to seamlessly translate between these PCS’s as needed. Two main methods are provided (in addition to color space access methods):

1. `ClccPCS::Check()` – This method checks to see if the current color space defined by the pixel is compatible to the source color space of the `ClccXform` object that will be using the pixel. It only makes conversions if the current color space is a PCS color space.
2. `ClccPCS::CheckLast()` – This method checks to see if the current color space defined by the pixel is compatible with the destination color space. It only makes conversions if the current color space is a PCS color space.

NOTE: This White Paper was updated in November 2025 to reference the `iccDEV` project.