Ghostscript 9.0 Color Management

Michael J. Vrhel, Ph.D.
Artifex Software
7 Mt. Lassen Drive, A-134
San Rafael, CA 94903, USA
www.artifex.com

Abstract

This document provides information about Ghostscript 9.0's color architecture. The document is suitable for users who wish to obtain accurate color with their output device as well as for developers who wish to customize Ghostscript to achieve a higher level of control and/or interface with a different color management module.

Revision 1.0

1 Introduction

As of release 9.0, the color architecture of Ghostscript has been significantly updated to be primarily based upon the ICC[1] format. Prior to this release, Ghostscript's color architecture was based heavily upon PostScript[2] Color Management (PCM). This is due to the fact that Ghostscript was designed prior to the ICC format and likely even before there was much thought about digital color management. At that point in time, color management was very much an art with someone adjusting controls to achieve the proper output color.

Today, almost all print color management is performed using ICC profiles as opposed to PCM. This fact along with the desire to create a faster, more flexible design was the motivation for the color architectural changes in release 9.0. Features of this new architecture include:

- Easy interface of different CMMs (Color Management Modules) with Ghostscript.
- Defining of all color spaces in terms of ICC profiles.
- Caching of linked transformations and internally generated profiles.
- Easily accessed manager for ICC profiles.
- Command line option for setting default profiles for Gray, RGB and CMYK source profiles.
- Command line option for setting device profile.
- Command line option for overriding input and output rendering intent.
- Command line option for specifying DeviceN profiles, named color structures, link profiles and proofing profiles.
- Command line option to override embedded profiles.
- Passing of object type along to CMM for creation of transforms that account for image, graphic or text.
- Efficient operation in multithreaded banded (c-list or display list) rendering.

The document is organized to first provide a higher level overview of the new ICC flow as well as how to make use of the new architecture. This is followed by details of the various functions and structures, which include the information necessary to interface other color management modules to Ghostscript as well as how to interface specialized color handling operations.

2 Overall Architecture and Typical Flow

Figure 1 provides a graphical overview of the various components that make up the architecture. The primary components are:

- The ICC Manager, which maintains the various default profiles.
- The Link Cache, which stores recently used linked transforms.
- The Profile Cache, which stores internally generated ICC profiles created from PostScript CIE based color spaces and CalRGB, CalGray PDF color spaces.
- The profiles contained in the root folder iccprofiles, which are used as default color spaces for the output device and for undefined source colors in the document.
- The color management module (CMM), which is the external engine that provides and performs the transformations (e.g. littleCMS).

In the typical flow, when a thread is ready to transform a buffer of data, it will request a linked transform from the Link Cache. When requesting a link, it is necessary to provide information to the CMM, which consists of a source color space, a destination color space, an object state (e.g. text, graphic, or image) and a rendering type (e.g. perceptual, saturation, colorimetric). The linked transform provides a mapping directly from the source color space to the destination color space. If a linked transform for these settings does not already exist in the Link Cache, a linked transform from the CMM will be obtained (assuming there is sufficient memory – if there is not sufficient memory then the requesting thread will need to wait). Depending upon the CMM, it is possible that the CMM may create a lazy linked object (i.e. create the real thing when it is asked to transform data). At some point, a linked transform will be returned to the requesting thread. The thread can then use this mapping to transform buffers of data through calls through an interface to the external CMM. Once the thread has completed its use of the link transform, it will notify the Link Cache. The Link Cache will then be able to release the link when it needs additional cache space due to other link requests.

3 PDL Color Definitions and ICC Profiles

To help reduce confusion, it is worthwhile to clarify terminology. In particular, the use of the terms process color and device color need to be defined in the context of ICC profiles. Both PDF[3] and PostScript (PS) have a distinction between process colors and device colors. In both PDF and PS, there is a conversion (e.g. via UCR/BG) from device colors to process colors. In an ICC work flow, the colors are transformed directly from an input color space

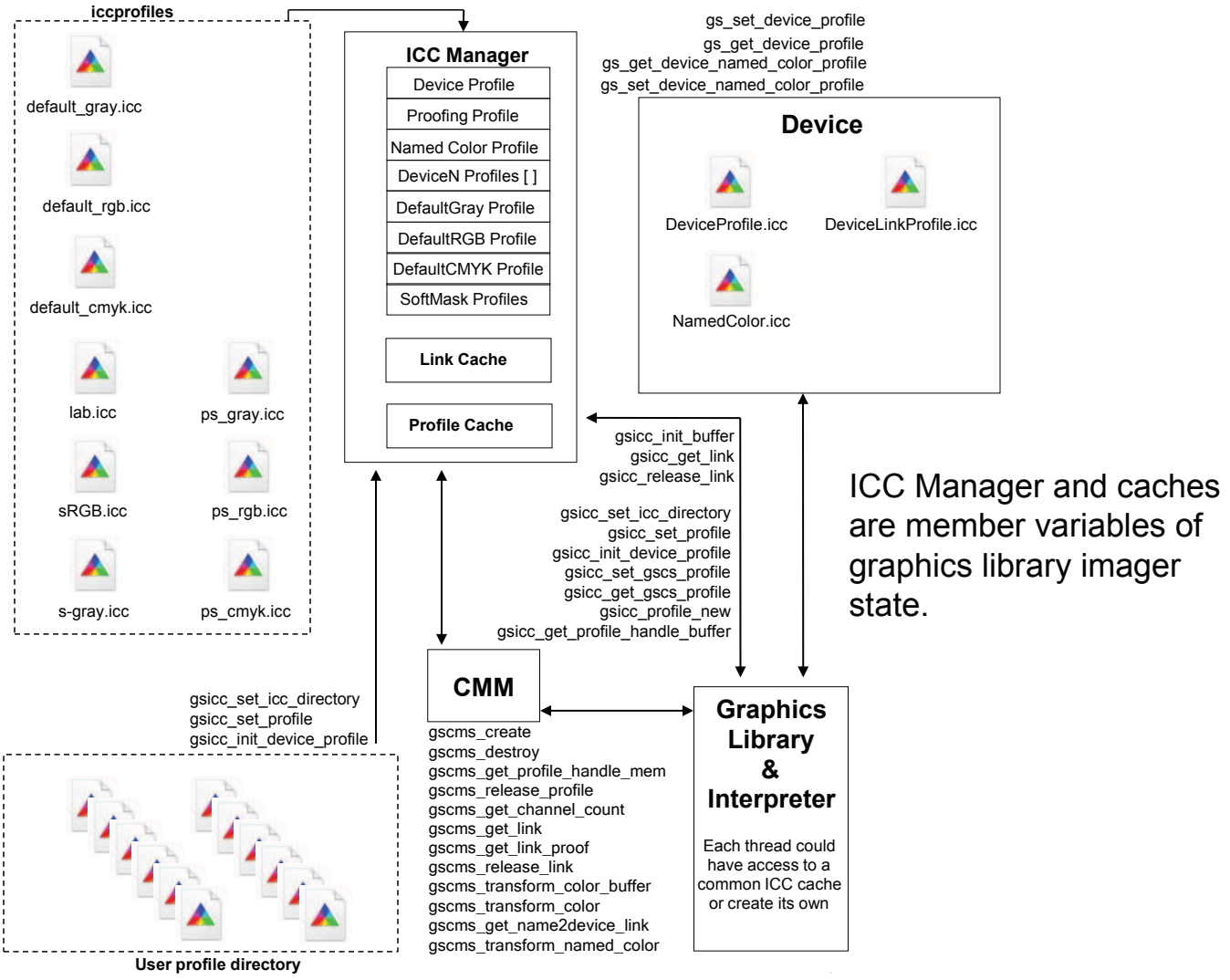

Figure 1: Graphical Overview of ICC Architecture

(often called the source space) to an output color space (often called the destination space). The output color space defined by the device's ICC profile is a mapping to what PDF and PS define as the process color space of the device. In other words, the "device color space" as defined by the device's ICC profile IS the process color space of PDF and PS. The ICC profile of the device is a mapping from a CIE color space to the process color space AND from the process color space to a CIE color space.

To understand this better, it may help to understand the method by which a print based ICC profile is created. To create an ICC profile for a device, a chart is printed using its process colors (e.g. CMYK). This chart is measured using a colorimeter or a spectrophotometer. This provides the forward mapping from process colors to CIELAB values. The inverse mapping (from CIELAB to process colors) is obtained by inverting this table usually through a brute force search and extrapolation method. These mappings are both packed into an ICC format, thereby defining mappings between the device "process colors" and the CIE color space.

4 Usage

The ICC branch introduces several new command line options that can be used for complete color management control. To define source colors that are not already colorimetrically defined in the source document, the following command line options can be invoked:

-sDefaultGrayProfile = my_gray_profile.icc

-sDefaultRGBProfile = my_rgb_profile.icc

-sDefaultCMYKProfile = my_cmyk_profile.icc

In this case, for example, any source gray colors will be interpreted as being defined by the ICC profile my_gray_profile.icc. If these profiles are not set, default ICC profiles will be used to define undefined colors. These default profiles are contained in the gs folder directory iccprofiles and are named default_gray.icc, default_rgb.icc and default_cmyk.icc. The profile default_gray.icc is defined to provide output along the neutral axis with an sRGB linearization. The profile default_rgb.icc is the V2 sRGB ICC profile and the profile default_cmyk.icc is a SWOP CMYK ICC profile.

In addition to being able to define undefined colors, it is possible to define the ICC profile for the output device using

-sOutputICCProfile = my_device_profile.icc

Care should be taken to make sure that the number of components associated with the

output device is the same as the number of components for the output device ICC profile (i.e. use an RGB profile for an RGB device). If the destination device is CMYK + SPOT colorants, then it is possible to specify either a CMYK ICC profile or an N-Color ICC profile for the device. If a CMYK profile is specified, then only the CMYK colorants will be color managed. If an output profile is not specified, then the default CMYK profile is used as the output profile.

A directory can be defined, which will be searched to find the above defined ICC profiles. This makes it easier for users who have their profiles contained in a single directory and do not wish to append the full path name in the above command line options. The directory is set using

-sICCProfilesDir = c:/my_icc_profiles

Note that if the build of gs or other PDL languages is performed with COMPILE_INITS=1, then the profiles contained in gs/iccprofiles will be placed in the ROM file system. If a directory is specified on the command line using -sICCProfilesDir=, that directory is searched before the iccprofiles/ directory of the ROM file system is searched.

Named color support for separation color spaces is specified through the command line option

-sNamedProfile = c:/my_namedcolor_stucture

While the ICC does define a named color format, the above structure can in practice be much more general for those who have more complex handling requirements of separation color spaces. For example, some developers wish to use their own proprietary-based format for spot color management. This command option is for developer use when an implementation for named color management is designed for the function **gsicc_transform_named_color** located in gsicc_cache.c . An example implementation is currently contained in the code [see comments above **gsicc_transform_named_color** in gsicc_cache.c]. For the general user, this command option should really not be used.

The above option deals with the handling of single spot colors. It is possible to specify ICC profiles or other structures for managing DeviceN source colors. This is done using the command line option

-sDeviceNProfile = c:/my_devicen_profile.icc

Note that neither PS nor PDF provide in-document ICC profile definitions for DeviceN color spaces. With this interface it is possible to provide this definition. The colorants tag order in the ICC profile defines the lay-down order of the inks associated with the profile. A windows-based tool for creating these source profiles is contained in gs/toolbin/color/icc_creator. If

non-ICC based color management of DeviceN source colors is desired by a developer, it is possible to use the same methods used for the handling of individual spot colors. In that case, a single proprietary structure could be used, which contains information about how to blend multiple colorants for accurate DeviceN color proofing.

The command line option

-sProofProfile = my_proof_profile.icc

enables the specification of a proofing profile, which will make the color management system link multiple profiles together to emulate the device defined by the proofing profile. This is currently under development and should be in-place before the official release of Ghostscript 9.0

The command line option

-sDeviceLinkProfile = my_link_profile.icc

makes it possible to include a device link profile in the color transformations. This is useful for devices that output raster content in a standard color space such as SWOP or Fogra CMYK, but it is desired to redirect this output to other CMYK devices. While it is possible to handle the color transformations in other manners (e.g. using a proofing profile) the use of device link profiles is not uncommon. The final linking profile is applied at the device level following rasterization to the destination color space as specified by the device profile. Ghostscript's rendering of the page description language PCL, which requires rendering into sRGB buffers, will make use of this device link profile to provide final conversion from sRGB to device CMYK prior to halftoning.

In some cases, it is desired to override any internal profiles that may exist within a document. The command line option

-dOverrideInternalProfiles = true

achieves this by replacing any document embedded ICC profiles with the Gray, RGB or CMYK default profile depending upon the channel count. Note that embedded CIELAB source profiles are obviously not overridden.

The following command specifies the rendering intent to use for the specified output profile.

-sOutputRenderingIntent = Perceptual

The default value is Perceptual. Other valid values include Saturation, Colorimetric and AbsoluteColorimetric.

Similarly, the desired rendering intents to use with Gray, RGB and CMYK input sources can be specified with the following commands.

-sInputGrayRenderIntent = Perceptual

-sInputRGBRenderIntent = Perceptual

-sInputCMYKRenderIntent = Perceptual

These will override any internal specification for rendering intents.

5 Overview of objects and methods

At this point, let us go into further detail of the architecture. Following this, we will discuss the requirements for interfacing another CMM to Ghostscript as well as where to interface in the new architecture for those who had made use of the now removed CUSTOM_COLOR_CALLBACK option in previous versions of Ghostscript.

5.1 ICC Manager

The ICC Manager is a reference counted member variable of Ghostscript's imager state. Its functions are to:

- Store the required profile information to use for Gray, RGB, and CMYK source colors that are NOT colorimetrically defined in the source document. These entries must always be set in the manager and are set to default values unless defined by the command line interface.
- Store the required profile information for the output device.
- Store the optional profile/structure information related to named colors and DeviceN colors.
- Store the proofing profile.
- Store the CIELAB source profile.
- Store the directory be used to search for ICC profiles specified for the above objects.
- Store settings for profile override, output rendering intent (i.e. perceptual, colorimetric, saturation or absolute colorimetric) and source color rendering intents.

- Store the profiles that are used for softmask rendering if soft masks are contained in the document.

The manager is created when the imaging state object is created for the graphics library. It is reference counted and allocated in garbage collected (GC) memory that is not stable with graphic state restores. The default gray, RGB and CMYK ICC color spaces as well as the device ICC color space are defined immediately during the initialization of the graphics library. If no ICC profiles are specified externally, then the ICC profiles that are contained in the root folder iccprofiles will be used. The ICC Manager is defined by the structure given below.

```
typedef struct gsicc_manager_s {
        cmm_profile_t *device_named;    /* The named color profile for the device */
        cmm_profile_t *default_gray;    /* Default gray profile for device gray */
        cmm_profile_t *default_rgb;     /* Default RGB profile for device RGB */
        cmm_profile_t *default_cmyk;    /* Default CMYK profile for device CMKY */
        cmm_profile_t *proof_profile;   /* Proofing profile */
        cmm_profile_t *output_link;     /* Output device Link profile */
        cmm_profile_t *device_profile;  /* The actual profile for the device */
        cmm_profile_t *lab_profile;     /* Colorspace type ICC profile from LAB to LAB */
        gsicc_devicen_t *device_n;      /* A linked list of profiles used for DeviceN support */
        gsicc_smask_t *smask_profiles;  /* Profiles used when we are in a softmask group */
        char *profiledir;               /* Directory used in searching for ICC profiles */
        uint namelen;
        gs_memory_t *memory;
        rc_header rc;
} gsicc_manager_t;
```

Operators that relate to the ICC Manager are contained in the file gsicc_manage.c/h and include the following:

int **gsicc_init_device_profile**(gs_state * pgs, gx_device * dev);

This initializes the device_profile member variable based upon the properties of the device. The device may have a profile defined in its dev→color_info.icc_profile member variable. If it does not, then a default profile will be assigned to the device.

int **gsicc_set_profile**(gsicc_manager_t *icc_manager, const char *pname, int namelen, gsicc_profile_t defaulttype);

This is used to set all the other profile related member variables in the ICC Manager. The member variable to set is specified by defaulttype.

void **gsicc_set_icc_directory**(const gs_imager_state *pis, const char* pname, int namelen);

This is used to set the directory for finding the ICC profiles specified by gsicc_set_profile.

gsicc_manager_t* **gsicc_manager_new**(gs_memory_t *memory);

Creator for the ICC Manager.

cmm_profile_t* **gsicc_profile_new**(stream *s, gs_memory_t *memory, const char* pname, int namelen);

Returns an ICC object given a stream pointer to the ICC content. The variables pname and namelen provide the filename and name length of the stream if it is to be created from a file. If the data is from the source stream, pname should be NULL and namelen should be zero.

int **gsicc_set_gscs_profile**(gs_color_space *pcs, cmm_profile_t *icc_profile, gs_memory_t * mem);

Sets the member variable cmm_icc_profile_data of the gs_color_space object (pointed to by pcs) to icc_profile.

cmm_profile_t* **gsicc_get_gscs_profile**(gs_color_space *gs_colorspace, gsicc_manager_t *icc_manager);

Returns the cmm_icc_profile_data member variable of the gs_color_space object.

gcmmhprofile_t **gsicc_get_profile_handle_buffer**(unsigned char *buffer, int profile_size);

Returns the CMS handle to the ICC profile contained in the buffer.

int **gsicc_init_iccmanager**(gs_state * pgs);

Initializes the ICC Manager with all the required default profiles.

void **gsicc_profile_serialize**(gsicc_serialized_profile_t *profile_data, cmm_profile_t *iccprofile);

A function used to serialize the icc profile information for embedding into the c-list.

cmm_profile_t* **gsicc_get_profile_handle_file**(const char* pname, int namelen, gs_memory_t *mem);

Given a profile file name, obtain a handle from the CMM.

void **gsicc_init_profile_info**(cmm_profile_t *profile);

With a profile handle already obtained from the CMM set up some of the member variables in the structure cmm_profile_t.

void **gsicc_init_hash_cs**(cmm_profile_t *picc_profile, gs_imager_state *pis);

Get the hash code for a profile.

gcmmhprofile_t **gsicc_get_profile_handle_clist**(cmm_profile_t *picc_profile, gs_memory_t *memory);

For a profile that is embedded inside the c-list, obtain a handle from the CMM.

gcmmhprofile_t **gsicc_get_profile_handle_buffer**(unsigned char *buffer, int profile_size);

For a profile that is contained in a memory buffer, obtain a handle from the CMM.

gsicc_smask_t* **gsicc_new_iccsmask**(gs_memory_t *memory);

Allocate space for the icc soft mask structure. Only invoked when softmask groups are used in rendering.

int **gsicc_initialize_iccsmask**(gsicc_manager_t *icc_manager);

Initialize the icc soft mask structure. Only invoked when softmask groups are used in rendering.

unsigned int **gsicc_getprofilesize**(unsigned char *buffer);

Get the size of a profile, as given by the profile information.

cmm_profile_t* **gsicc_read_serial_icc**(gx_device * dev, int64_t icc_hashcode);

Read out the serialized icc data contained in the clist for a given hash code.

cmm_profile_t* **gsicc_finddevicen**(const gs_color_space *pcs, gsicc_manager_t *icc_manager);

Search the DeviceN profile array for a profile that has the same colorants as the DeviceN color space in the PDF or PS document.

gs_color_space_index **gsicc_get_default_type**(cmm_profile_t *profile_data);

Detect profiles that were set as part of the default settings. These are needed to differentiate between embedded document icc profiles and ones that were supplied to undefined device source colors (e.g. DeviceRGB). During high level device writing (e.g. pdfwrite), these default profiles are usually NOT written out.

void **gsicc_profile_reference**(cmm_profile_t *icc_profile, int delta);

Enable other language interpreters (e.g. gxps) to adjust the reference count of a profile.

int **gsicc_getsrc_channel_count**(cmm_profile_t *icc_profile);

Returns the number of device channels for a profile.

5.2 Link Cache

The Link Cache is a reference counted member variable of Ghostscript's imager state. Its function is to maintain a cache of recently used links that had been provided by the CMM. The Link Cache is designed with semaphore calls to allow multi-threaded c-list (display list) rendering to share a common cache.

The Link Cache is allocated in stable GC memory. Operators that relate to the Link Cache are contained in the file gsicc_cache.c/h and include the following:

gsicc_link_cache_t* **gsicc_cache_new**(gs_memory_t *memory);

Creator for the Link Cache.

void **gsicc_init_buffer**(gsicc_bufferdesc_t *buffer_desc, unsigned char num_chan, unsigned char bytes_per_chan, bool has_alpha, bool alpha_first, bool is_planar, int plane_stride, int row_stride, int num_rows, int pixels_per_row);

This is used to initialize a gsicc_bufferdesc_t object. Two of these objects are used to describe the format of the buffers that are used in transforming color data.

gsicc_link_t* **gsicc_get_link**(gs_imager_state * pis, gs_color_space *input_colorspace, gs_color_space *output_colorspace, gsicc_rendering_param_t *rendering_params, gs_memory_t *memory, bool include_softproof);

This returns the link given the input color space, the output color space, and the rendering intent. When the requester of the link is finished using the link, it should release the link. When a link request is made, the Link Cache will use the parameters to compute a hash code. This hash code is used to determine if there is already a link transform that meets the needs of the request. If there is not a link present, the Link Cache will obtain a new one from the CMM (assuming there is sufficient memory), updating the cache.

The linked hash code is a unique code that identifies the link for an input color space, an object type, a rendering intent and an output color space. The operation that does the merging of these four pieces of information can easily be altered to ignore object type and/or rendering intent if so desired.

Note, that the output color space can be different than the device space. This occurs for example, when we have a transparency blending color space that is different than the device color space.

gsicc_link_t* **gsicc_get_link_profile**(gs_imager_state *pis, cmm_profile_t *gs_input_profile, cmm_profile_t *gs_output_profile, gsicc_rendering_param_t *rendering_params, gs_memory_t *memory, bool include_softproof);

This is similar to the above operation **gsicc_get_link** but will obtain the link with profiles that are not member variables of the gs_color_space object.

void **gsicc_get_icc_buff_hash**(unsigned char *buffer, int64_t *hash, unsigned int buff_size);

This computes the hash code for the buffer that contains the ICC profile.

int **gsicc_transform_named_color**(float tint_value, byte *color_name, uint name_size, gx_color_value device_values[], const gs_imager_state *pis, cmm_profile_t *gs_output_profile, gsicc_rendering_param_t *rendering_params, bool include_softproof);

This performs a transformation on the named color given a particular tint value return device_values.

void **gsicc_release_link**(gsicc_link_t *icclink);

This is called to notify the cache that the requester for the link no longer needs it. The link is reference counted, so that the cache knows when it is able to destroy the link. The link is released through a call to the CMM.

5.3 Interface of Ghostscript to CMM

Ghostscript interfaces to the CMM through a single file. The file gsicc_littlecms.c/h is a reference interface between littleCMS and Ghostscript. If a new library is used (for example, if littleCMS is replaced with Windows ICM on a Windows platform (giving Windows color system (WCS) access on Vista or System 7)), the interface of these functions will remain the same, but internally they will need to be changed. Specifically, the functions are as follows:

void **gscms_create**(void **contextptr);

This operation performs any initializations required for the CMM.

void **gscms_destroy**(void **contextptr);

This operation performs any cleanup required for the CMM.

gcmmhprofile_t **gscms_get_profile_handle_mem**(unsigned char *buffer, unsigned int input_size);

This returns a profile handle for the profile contained in the specified buffer.

void **gscms_release_profile**(void *profile);

When a color space is removed or we are ending, this is used to have the CMM release a profile handle it has created.

int **gscms_get_input_channel_count**(gcmmhprofile_t profile);

Provides the number of colorants associated with the ICC profile. Note that if this is a device link profile this is the number of input channels for the profile.

int **gscms_get_output_channel_count**(gcmmhprofile_t profile);

If this is a device link profile, then the function returns the number of output channels for the profile. If it is a profile with a PCS, then the function should return a value of three.

gcmmhlink_t **gscms_get_link**(gcmmhprofile_t lcms_srchandle, gcmmhprofile_t lcms_deshandle, gsicc_rendering_param_t *rendering_params);

This is the function that obtains the linkhandle from the CMM. The call **gscms_get_link** is usually called from the Link Cache. In the graphics library, calls are made to obtain links using **gsicc_get_link**, since the link may already be available. However, it is possible to use **gscms_get_link** to obtain linked transforms outside the graphics library. For example, this is the case with the XPS interpreter, where minor color management needs to occur to properly handle gradient stops.

gcmmhlink_t **gscms_get_link_proof**(gcmmhprofile_t lcms_srchandle, gcmmhprofile_t lcms_deshandle, gcmmhprofile_t lcms_proofhandle, gsicc_rendering_param_t *rendering_params);

This function is similar to the above function but includes a proofing ICC profile. If the proofing profile is defined, then the output should appear as if it were printed on the device defined by the proofing profile.

void **gscms_release_link**(gsicc_link_t *icclink);

When a link is removed from the cache or we are ending, this is used to have the CMM release the link handles it has created.

void **gscms_transform_color_buffer**(gsicc_link_t *icclink, gsicc_bufferdesc_t *input_buff_desc, gsicc_bufferdesc_t *output_buff_desc, void *inputbuffer, void *outputbuffer);

This is the function through which all color transformations will occur if they are to go through the CMM. This function will be called in the code anytime that we are transforming color from the current graphic state color to the Output Device color space or to the Blending Color Space, or out of the Blending color space to the Color Space of the parent layer in the transparency stack. Note that if the source hash code and the destination hash code are the same, the transformation will not occur as the source and destination color spaces are identical. This feature can be used to enable "device colors" to pass unmolested through the color processing.

void **gscms_transform_color**(gsicc_link_t *icclink, void *inputcolor, void *outputcolor, int num_bytes, void **contextptr);

This is a special case where we desire to transform a single color. While it would be possible to use **gscms_transform_color_buffer** for this operation, single color transformations are frequently required and it is possible that the CMM may have special optimized code for this operation.

int **gscms_transform_named_color**(gsicc_link_t *icclink, float tint_value, const char* ColorName, gx_color_value device_values[]);

Get a device value for the named color. While there exist named color ICC profiles and littleCMS supports them, the code in gsicc_littlecms.c is not designed to use that format. The named color object need not be an ICC named color profile but can be a proprietary type table. This is discussed further where -sNamedProfile is defined in the Usage section.

void **gscms_get_name2device_link**(gsicc_link_t *icclink, gcmmhprofile_t lcms_srchandle, gcmmhprofile_t lcms_deshandle, gcmmhprofile_t lcms_proofhandle, gsicc_rendering_param_t *rendering_params, gsicc_manager_t *icc_manager);

This is the companion operator to **gscms_transform_named_color** in that it provides the link transform that should be used when transforming named colors when named color ICC profiles are used for named color management. Since **gscms_transform_named_color** currently is set up to use a non-ICC table format, this function is not used.

gcmmhprofile_t **gscms_get_profile_handle_file**(const char *filename);

Obtain a profile handle given a file name.

char* **gscms_get_clrtname**(gcmmhprofile_t profile, int k);

Obtain the *k*th colorant name in a profile. Used for DeviceN color management with ICC profiles.

int **gscms_get_numberclrtnames**(gcmmhprofile_t profile);

Return the number of colorant names that are contained within the profile. Used for DeviceN color management with ICC profiles.

gsicc_colorbuffer_t **gscms_get_profile_data_space**(gcmmhprofile_t profile);

Get the color space type associated with the profile.

6 PDF and PS CIE color space handling

If a color space is a PDF or PostScript (PS) CIE color space type (other than ICC), these color spaces are converted to appropriate ICC objects. The profiles are created by the functions in gsicc_create.c. Since this file is only needed by the PS and PDF interpreter, it is contained in the psi subdirectory of Ghostscript's folder tree and is not needed for PCL or XPS builds.

Performing this conversion, enables the ICC based CMM full control over all color management. To avoid frequent conversions due to frequent color space changes, these color spaces are cached in the profile cache and indexed using a value related to their resource IDs. This is the profile cache object that is indicated in Figure 1. In PDF, it is possible to define CIELAB color values directly. The ICC profile lab.icc contained in iccprofiles of Figure 1 is used as the source ICC profile for color defined in this manner.

Currently PostScript color rendering dictionaries (CRDs) are ignored if defined in the current code. Instead, a device ICC profile should be used to define the color for the output device. An upcoming change will be to convert CRDs to equivalent ICC profiles, which will then work with the existing workflow.

Note that if littleCMS is replaced, gsicc_create.c still requires icc34.h, since it uses the type definitions in that file in creating the ICC profiles from the PS and PDF CIE color spaces.

7 Device Interaction

From Figure 1, it is clear that the device can communicate to the graphics library its ICC profiles. Depending upon the settings of the device (e.g. paper type, ink, driver settings) it may provide a different profile as well as indicate a desired rendering intent. Unless overridden by command line arguments, this information will be used to populate the ICC manager's Device Profile. Currently, this interaction is under development and should be in place with the official release of Ghostscript 9.0.

8 DeviceN and Separation colors

DeviceN and Separation colors are handled differently depending upon the source PDL that is being processed. In Microsoft's XPS format, all input DeviceN or Separation type colors are required to have an associated ICC profile. If one is not provided, then per the XPS specification[4] a SWOP CMYK profile is assumed for the first four colorants and the remaining colorants are ignored. With PDF DeviceN or Separation colors, the document defines a tint transform and an alternate color space, which could be any of the CIE (e.g. CalGray, CalRGB, Lab, ICC) or device (e.g. Gray, RGB, CMYK) color spaces. If the input source document is PDF or PS and the output device does not understand the colorants defined in the DeviceN color space, then the colors will be transformed to the alternate color space and color managed from there.

For cases when the device **does** understand the spot colorants of the DeviceN color space, the preferred handling of DeviceN varies. Many prefer to color manage the CMYK components with a defined CMYK profile, while the other spot colorants pass through unmolested. This will be the default manner by which Ghostscript will handle DeviceN input colors. In other words, if the device profile is set to a particular CMYK profile, and the output device is a separation device, which can handle all spot colors, then the CMYK process colorants will be color managed, but the other colorants will not be managed. If it is desired that the CMYK colorants not be altered also, it will be possible to achieve this by having the source and destination ICC profiles the same. This will result in an identity transform, which will not be used when processing the CMYK colorants.

It should be noted that an ICC profile can define color spaces with up to 15 colorants. For a device that has 15 or fewer colorants, it is possible to provide an ICC profile for such a device. In this case, all the colorants will be color managed through the ICC profile. For cases beyond 15, the device will be doing direct printing of the DeviceN colors outside of the 15 colorants.

9 PCL and XPS Support

PCL[5] makes use of the new color management architecture primarily in its output devices. Source colors are specified to be in the sRGB color space. If the commands in the PCL file require rendering into an RGB buffer due to blending of transparency operations, then these buffers will be converted to the appropriate CMYK color space using the CMM, when the drawing commands have completed in that region. If the commands do not require rendering into a continuous tone RGB buffer, then the conversion from RGB to CMYK will occur prior to rendering into a halftone CMYK buffer.

Full ICC support for XPS[4] is now contained in ghostxps. This includes the handling of profiles for DeviceN color spaces, Named colors and for profiles embedded within images.

10 CUSTOM_COLOR_CALLBACK developers

In earlier versions of Ghostscript, there existed a compile define named CUSTOM_COLOR_CALLBACK, which provided developers with a method to intercept color conversions and provide customized processing in particular for Separation and DeviceN input color spaces. Using specialized mixing models in place of the standard tint transforms, accurate proofing of the spot colorants was obtainable. An interface for custom handling of separation colors is now performed by customization of the function gsicc_transform_named_color. An example, implementation is currently in place, which uses a look-up-table based upon the colorant name. The look-up-table is stored in the device_named object of the icc manager. The structure can be stored in the location using -sNamedProfile = c:/my_namedcolor_stucture.

DeviceN color handling is defined by an object stored in the device_n entry of the icc_manager. Currently, the example implementation is to use an array of ICC profiles that describe the mixing of the DeviceN colors of interest. This array of profiles is contained in the device_n entry of the icc_manager. In this case, a multi-dimensional look-up-table is essentially used to map the overlayed DeviceN colors to the output device colorants.

If a mathematical mixing model is to be used for the DeviceN colors instead of an ICC-based approach, it will be necessary to store the data required for mixing either in the device_n entry or, if the same data is used for separation colors, the data in the named_color location can be used. In either case, a single line change will be required in **gx_install_DeviceN** where a call is currently made to **gsicc_finddevicen** to locate a matching DeviceN ICC profile for DeviceN color management. In place of this call, it will be necessary to make a call to a function that will prepare an object that can map colors in this DeviceN space to the real device values. A pointer to this object is then returned by the function. If the colorants cannot be handled, the function should return NULL. If the function can handle the colorants, then when the link request is made between this color space and the output device profile with the function **gsicc_get_link** it will be necessary to detect

that the source object is not a standard ICC profile but a special customized object. This may require the addition of a special flag in the cmm_profile_t structure. This flag would then be checked to determine if a call should be made to the CMM or to a custom color management function that makes use of the pointer previously obtained from **gsicc_finddevicen**. When the call is made to apply the transformation using **gscms_transform_color_buffer** or **gscms_transform_color**, the appropriate operation should be applied to the incoming data in place of the standard call to the CMM. Artifex will be able to assist those developers who need help in the transition from the previous CUSTOM_COLOR_CALLBACK architecture to the new design.

References

[1] Specification ICC.1:2004-10 (Profile version 4.2.0.0) Image technology colour management - Architecture, profile format, and data structure. (http://www.color.org/ICC1v42_2006-05.pdf), Oct. 2004.

[2] PostScript® Language Reference Third Edition, Adobe Systems Incorporated, Addison-Wesley Publishing, (http://partners.adobe.com/public/developer/ps/index_specs.html) Reading Massachusetts, 1999.

[3] PDF Reference Sixth Edition Ver. 1.7, Adobe Systems Incorporated, (http://www.adobe.com/devnet/pdf/pdf_reference.html), November 2006.

[4] XML Paper Specification Ver. 1.0, Microsoft Corporation, (http://www.microsoft.com/whdc/xps/xpsspec.mspx), 2006.

[5] PCL5 Printer Language Technical Reference Manual, Hewlett Packard, (http://h20000.www2.hp.com/bc/docs/support/SupportManual/bpl13210/bpl13210.pdf), October 1992.