Ghostscript 9.05 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 the color architecture in Ghostscript 9.05. 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 1.1

1 Introduction

With release 9.0, the color architecture of Ghostscript was updated to primarily use the ICC[1] format for its color management needs. 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. Since 9.0, several new features and capabilities have been added. As of the 9.05 release, features of the color architecture include:

- Easy to interface different CMMs (Color Management Modules) with Ghostscript.
- ALL color spaces are defined in terms of ICC profiles.
- Linked transformations and internally generated profiles are cached.
- Easily accessed manager for ICC profiles.
- Easy to specify default profiles for DeviceGray, DeviceRGB and DeviceCMYK color spaces.
- Devices can readily communicate their ICC profiles and have their ICC profiles set.
- Operates efficiently in a multithreaded environment.
- Handles named colors (spots) with ICC named color profile or proprietary format.
- ICC color management of Device-N colors or alternatively customizable spot color handing.
- Includes object type (e.g. image, graphic, text) and rendering intent into the computation of the linked transform.
- Ability to override document embedded ICC profiles with Ghostscript's default ICC profiles.
- Easy to specify unique **source** ICC profiles to use with CMYK and RGB graphic, image and text objects.
- Easy to specify unique **destination** ICC profiles to use with graphic, image and text objects.

- Easy to specify different rendering intents (perceptual, colorimetric, saturation, absolute colorimetric) for graphic, image and text objects.
- Control to force gray source colors to black ink only when rendering to output devices that support black ink.

The document is organized to first provide a high level overview of the 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 engine that provides and performs the transformations (e.g. little CMS).
- The profiles associated with the device, which include profiles dependent upon object type, a proofing profile and a device link profile.

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

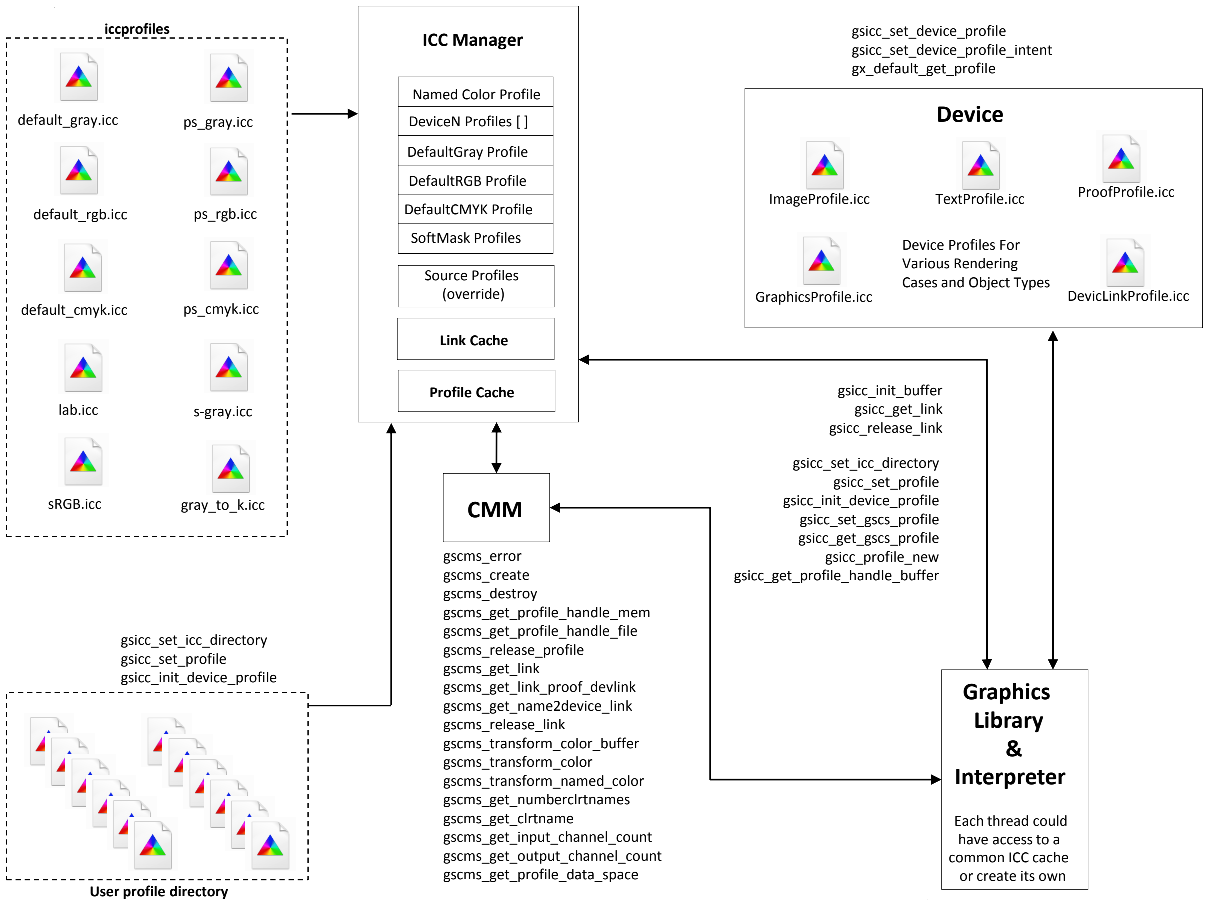


Figure 1: Graphical Overview of Ghostscript's Color Architecture

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

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 (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

There are a number of command line options available for color control. These options are also available as device parameters and so can be set from Ghostscript's command prompt when Ghostscript is used in "server-mode" operation.

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 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.

It is possible to have Ghostscript use the above specified ICC profiles in place of ICC profiles embedded in the document. This is achieved using

-dOverrideICC = true/false

which, when set to true overrides any ICC profiles contained in the source document with the profiles specified by sDefaultGrayProfile, sDefaultRGBProfile, sDefaultCMYKProfile. Note that if no profiles are specified for the default Device color spaces, then the system default profiles will be used. For detailed override control in the specification of source colors see SourceObjectICC.

In addition to being able to define undefined source 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.

If an N-Color (NCLR) ICC profile is specified for the output device (valid for tiffsep and psdcmyk devices), then it is necessary to specify the name of the colorants in the profile. This specification is done using

-sICCOutputColors="Cyan, Magenta, Yellow, Black, Orange, Violet"

Where the colorants listed are shown as an example. The list of the colorant names must be in the order that they exist in the profile. Note that if a colorant name that is specified for the profile occurs also within the document (e.g. "Orange" above), then these colorants will be associated with the same separation. It is possible through a compile time option LIMIT_TO_ICC defined in gdevdevn.h to restrict the output colorants of the psdcmyk and tiffsep device to the colorants of the ICC profile or to allow additional spot colorants in the document to be created as different separations. If restricted, the other spot colorants will go through the alternate tint transform and then be mapped to the color space defined by the N-CLR 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_structure
```

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 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. This would require the addition of code in **gx_concretize_DeviceN** similar to what is done in **gx_concretize_Separation** (with the call of **gsicc_transform_named_color**) for the specialized handing of spot colors described above.

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. See Section 4.2 for details on this option.

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 work flows where one wants to map colors first to a standard color space such as SWOP or Fogra CMYK, but it is desired to redirect this output to other CMYK devices. See Section 4.2 for details on this option.

It is possible for a document to specify the rendering intent to be used when performing a color transformation. Ghostscript is set up to handle four rendering intents with the nomenclature of Perceptual, Colorimetric, Saturation, and Absolute Colorimetric, which matches the terminology used by the ICC format. By default, per the specification, the rendering intent is Perceptual for PDF and PS documents. In many cases, it may be desired to ignore the source settings for rendering intent. This is achieved through the use of two parameter settings which are

-dOverrideRI = true/false

which, when set to true overrides the rendering intent contained in the source document with the rendering intent that has been specified by

-dRenderIntent = intent

which sets the rendering intent that should be used with the profile specified above by -sOutputICCProfile. The options for intent are 0, 1, 2 and 3, which correspond to the ICC intents of Perceptual, Colorimetric, Saturation, and Absolute Colorimetric.

There are two additional special color handling options that may be of interest to some users. One is

-dDeviceGrayToK = true/false

By default, Ghostscript will map DeviceGray color spaces to pure K when the output device is CMYK based. This may not always be desired. In particular, it may be desired to map from the gray ICC profile specified by -sDefaultGrayProfile to the output device profile. To achieve this, one should specify -dDeviceGrayToK=false. The gray_to_k.icc profile in ./profiles is used to achieve this mapping of source gray to the colorant K.

In certain cases, it may be desired to **not** perform ICC color management on DeviceGray, DeviceRGB and DeviceCMYK source colors. This can occur in particular. if one is attempting to create an ICC profile for a target device and needed to print pure colorants. In this case, one may want instead to use the traditional Postscript 255 minus operations to convert between RGB and CMYK with black generation and undercolor removal mappings. To achieve these types of color mappings use the following command set to true

-dUseFastColor = true/false

4.1 Object dependent color management

It is often desired to perform unique mappings based upon object types. For example, one may want to perform one color transformation on text colors to ensure a black text and a different transformation on image colors to ensure perceptually pleasing images and yet another transformation on graphics to create saturated colors. To achieve this, Ghostscript provides a unprecedented amount of color control based upon object type.

The following commands, enable one to specify unique **output** ICC profiles and rendering intents for text, graphic and image objects. As shown in Figure 1, these profiles are stored in the device structure. Specifically, the command options are:

-sGraphicICCProfile = filename

Sets the ICC profile that will be associated with the output device for vector-based graphics (e.g. solid color Fill, Stroke operations). This option can be used to obtain more saturated colors for graphics. Care should be taken to ensure that the number of colorants associated with the device is the same as the profile.

-sGraphicIntent = intent

Sets the rendering intent that should be used with the profile specified above by -sGraphicICCProfile. The options are the same as specified for -dRenderIntent. It is also necessary to set -

dOverrideRI=true.

-sImageICCProfile = filename

Sets the ICC profile that will be associated with the output device for images. This can be used to obtain perceptually pleasing images. Care should be taken to ensure that the number of colorants associated with the device is the same as the profile.

-sImageIntent = intent

Sets the rendering intent that should be used with the profile specified above by -sImageICCProfile. The options are the same as specified for -dRenderIntent. It is also necessary to set -dOverrideRI=true.

-sTextICCProfile = filename

Sets the ICC profile that will be associated with the output device for text. This can be used ensure K only text at the output. Care should be taken to ensure that the number of colorants associated with the device is the same as the profile.

-sTextIntent = intent

Sets the rendering intent that should be used with the profile specified above by -sTextICCProfile. The options are the same as specified for -dRenderIntent. It is also necessary to set -dOverrideRI=true.

In addition to being able to have the output ICC profile dependent upon object type, it is possible to have the **source** ICC profile and rendering intents be dependent upon object types for RGB and CMYK objects. Because this requires the specification of 12 new parameters and is only used in specialized situations, the specification is made through a single text file. The text file is specified to Ghostscript using

-sSourceObjectICC = filename

This option provides an extreme level of override control to specify the source color spaces and rendering intents to use with graphics, images and text for both RGB and CMYK source objects. The specification is made through a file that contains on a line, a key name to specify the object type (e.g. Image_CMYK) followed by an ICC profile file name and a rendering intent number (0 for perceptual, 1 for colorimetric, 2 for saturation, 3 for absolute colorimetric). An example file is given in ./gs/toolbin/color/src_color/objsrc_profiles_example.txt.

Profiles to demonstrate this method of specification are also included in this folder. Note that if objects are colorimetrically specified through this mechanism, other operations like -sImageIntent, -dOverrideICC, have no affect.

The example file mentioned above contains the following tab delimited lines

Graphic_CMYK	cmyk_src_cyan.icc	0
Image_CMYK	cmyk_src_magenta.icc	0
Text_CMYK	cmyk_src_yellow.icc	0
Graphic_RGB	rgb_source_red.icc	0
Image_RGB	rgb_source_green.icc	0
Text_RGB	rgb_source_blue.icc	0

where the first item in the line is the key word, the second item in the line is the file name of the **source** ICC profile to use for that object type and the third item specifies the rendering intent. Note that not all types need to be specified. It is possible to have only a single type specified in the file (e.g. Image_CMYK). The other items would render in a normal default fashion in this case.

For those interested in this level of control, it is recommended to execute a number of examples. In the first example, copy the files in ./gs/toolbin/color/src_color/ to ./iccprofiles and render the file ./examples/text_graph_image_cmyk_rgb.pdf with the option -sSourceObjectICC = objsrc_profiles_example.txt to an RGB device (e.g. tiff24nc). Note, to ensure that Ghostscript can find all the files and to avoid having to do a full rebuild to create the ROM file system, you may want to specify the icc directory using
-sICCProfilesDir="your_full_path_to_iccprofiles/", which provides the full path to ./iccprofiles/. Windows users should be sure to use the forward slash delimiter due to the special interpretation of "\" by the Microsoft C startup code.

Figure 2 displays the source file text_graph_image_cmyk_rgb.pdf rendered with default settings and Figure 3a displays the result when rendered using -sSourceObjectICC = objsrc_profiles_example.txt. The profiles specified in objsrc_profiles_example.txt are designed to render object types to the color specified in their name when used as a source profile. In this case, RGB graphics, images and text are rendered red, green and blue respectively and CMYK graphics, images and text are rendered cyan, magenta and yellow respectively.

Modifying the contents of the objsrc_profiles_example.txt file to

Graphic_CMYK	cmyk_src_renderintent.icc	0
Image_CMYK	cmyk_src_renderintent.icc	1
Text_CMYK	cmyk_src_renderintent.icc	2

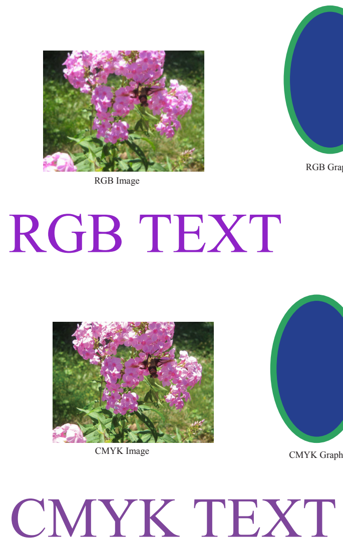


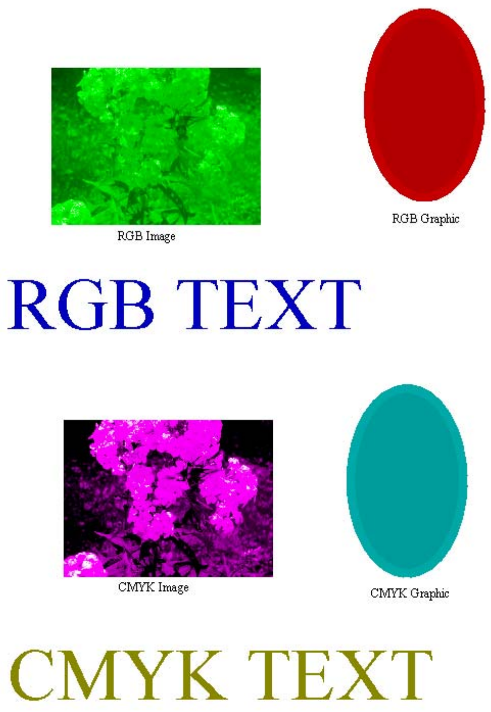
Figure 2: Example file with mixed content. The file includes RGB and CMYK text, graphics, and iamges

and rendering the file ./examples/text_graph_image_cmyk_rgb.pdf to an RGB device, one obtains the output shown in Figure 3b. In this case, we demonstrated the control of rendering intent based upon object type. The profile cmyk_src_renderintent.icc is designed to create significantly different colors for its different intents. Since we only specified this for the CMYK objects we see that they are the only objects effected and that this profile renders its perceptual intent cyan, its colorimetric intent magenta and its saturation intent yellow.

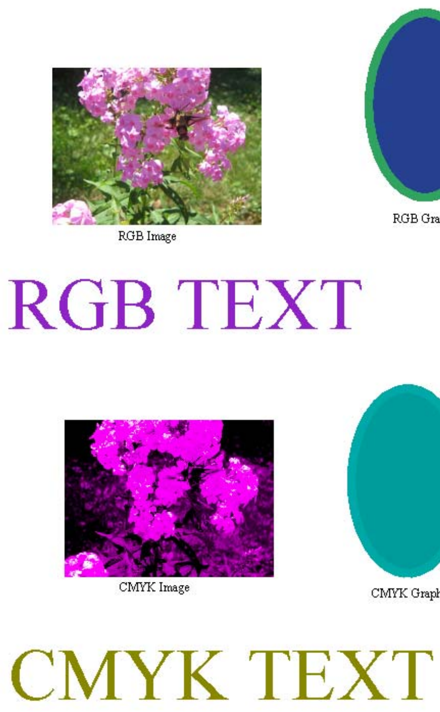
For another example of object dependent color management, copy the files in ./toolbin/color/icc_creator/effects to ./iccprofiles. Now specify unique output ICC profiles for different object types using the command line options

```
-sGraphicICCProfile = yellow_output.icc
-sImageICCProfile = magenta_output.icc
-sTextICCProfile = cyan_output.icc
```

while rendering the file text_graph_image_cmyk_rgb.pdf to a CMYK device (e.g. tiff32nc). Figure 4a displays the results. In this case, the profiles, cyan_output.icc, yellow_output.icc and magenta_output.icc render a color that is indicated by their name when used as an output profile.

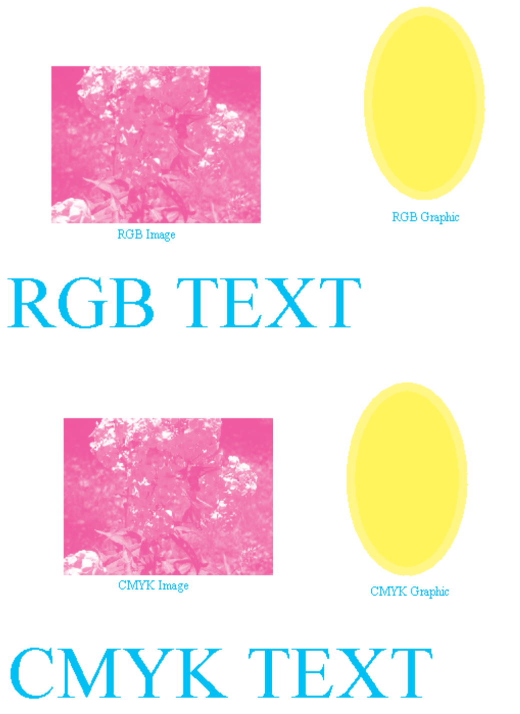


(a) Source profiles vary with object type

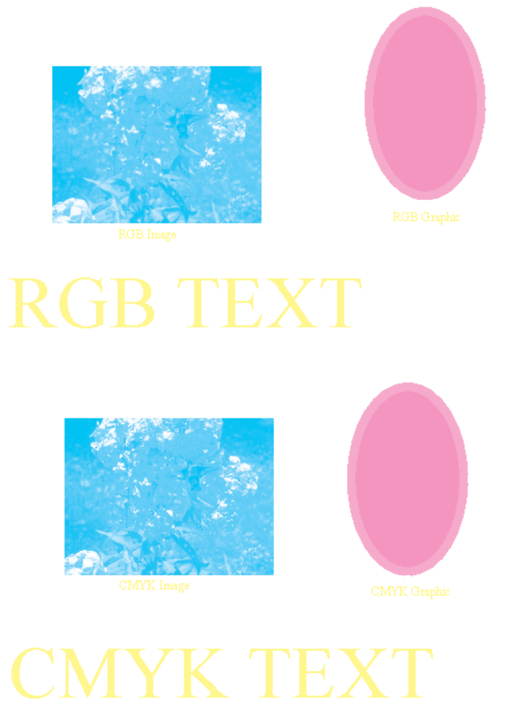


(b) Rendering intents vary with CMYK source object type

Figure 3: Examples of object based color transformations for the file from Figure 2 by specifying **source** profiles and/or rendering intents



(a) Destination profiles vary with object type



(b) Destination intents vary with object type

Figure 4: Examples of object based color transformations for the file from Figure 2 by specifying **destination** profiles and/or intents

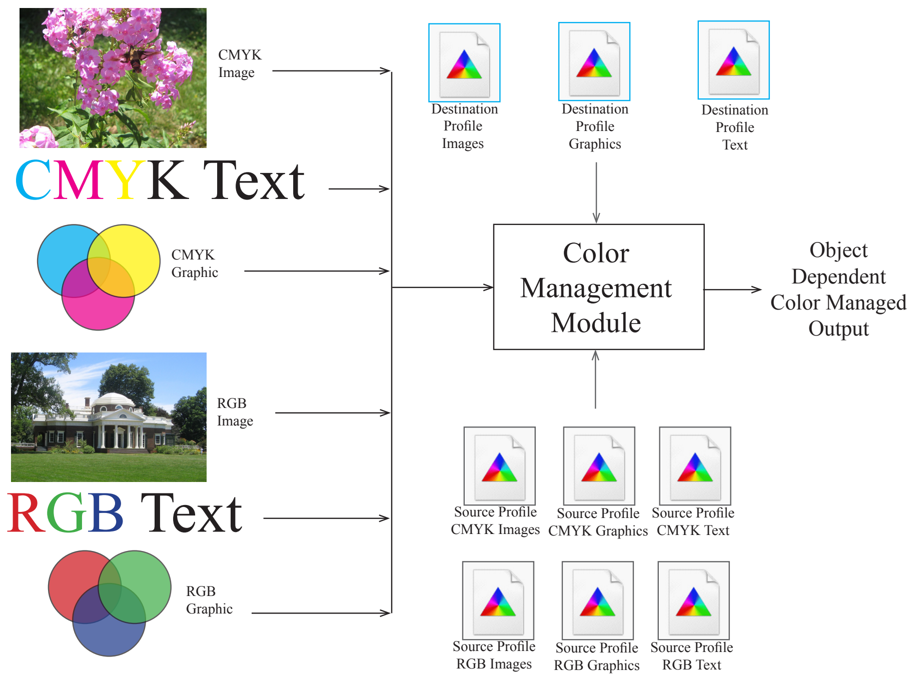

CMYK Image
CMYK Text
CMYK Graphic
RGB Image
RGB Text
RGB Graphic
Destination Profile Images
Destination Profile Graphics
Destination Profile Text
Color Management Module
Object Dependent Color Managed Output
Source Profile CMYK Images
Source Profile CMYK Graphics
Source Profile CMYK Text
Source Profile RGB Images
Source Profile RGB Graphics
Source Profile RGB Text

Figure 5: Overview of profiles that can be used in object dependent color management

Finally, in yet another example, we can demonstrate the effect of rendering intent for different objects using the command line options

-sGraphicICCProfile = cmyk_des_renderintent.icc
-sImageICCProfile = cmyk_des_renderintent.icc
-sTextICCProfile = cmyk_des_renderintent.icc
-dImageIntent = 0
-dGraphicIntent = 1
-dTextIntent = 2
-dOverrideRI

Figure 4b displays the result. The profile cmyk_des_renderintent.icc is designed such that the perceptual rendering intent outputs cyan only, the colorimetric intent outputs magenta only and the saturation intent outputs yellow only.

A graphical overview of the object dependent color control is shown in Figure 5, which shows how both the source and/or the destination ICC profiles can be specified.

Finally, it should be mentioned that Ghostscript has the capability to maintain object type information even through transparency blending. This is achieved through the use of a special tag plane during the blending of the objects. When the final blending of the objects occurs this tag information is available. Mixed objects will be indicated as such (e.g text blended with image). A device can have a specialized put_image operation that can handle the pixel level color management operation and apply the desired color mapping for different blend cases. The bittagrgb device in Ghostscript provides a demonstration of the use of the tag information.

4.2 Proof and Device-Link Profiles

As shown in Figure 1, the proofing profile and the device link profile are associated with the device. If these profiles have been specified using the options -sProofProfile = my_proof_profile.icc and -sDeviceLinkProfile = my_link_profile.icc, then when the graphics library maps a source color defined by the ICC profile source.icc to the device color values, a transformation is computed by the CMM that consists of the steps shown in Figure 6. In this Figure, Device ICC Profile is the ICC profile specified for the actual device (this can be specified using -sOutputICCProfile). In practice, the CMM will create a single mapping that performs the transformation of the multiple mappings shown in Figure 6. If we specify a proofing profile, then our output should provide a proof of how the output would appear if it had been displayed or printed on the proofing device defined by the proofing profile. The device link profile is useful for cases where one may have a work flow that consists of always rendering to a common CMYK space such as Fogra 39 followed by a mapping with a specialized device link profile. In this case, the profile specified by -sOutputICCProfile would be the profile for the common CMYK space.

5 Details 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 details for customization of handling Separation and DeviceN color spaces.

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

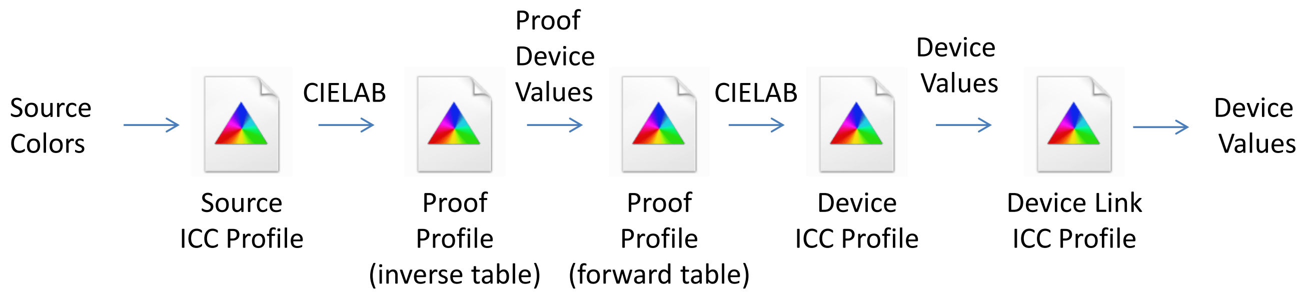


Figure 6: Flow of data through source, proof, destination and device link ICC profiles

always be set in the manager and are set to default values unless defined by the command line interface.

- Store the optional profile/structure information related to named colors and DeviceN colors.
- Store the CIELAB source profile.
- Store the specialized profile for mapping gray source colors to K-only CMYK values.
- 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.
- Store the profiles used for object dependent source color specification through the use of -sSourceObjectICC.
- Store the boolean flags for profile and rendering intent override of source settings.

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 stable with graphic state restores. The default gray, RGB and CMYK ICC color spaces 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 *lab_profile;               /* Colorspace type ICC profile from LAB to LAB */
        cmm_profile_t *graytok_profile;           /* A specialized profile for mapping gray to K */
        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 */
        bool override_internal;                   /* Override source ICC profiles */
        bool override_ri;                         /* Override source rendering intent */
        cmm_srcgtag_profile_t *srcgtag_profile;   /* Object dependent source profiles */
        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:

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

Creator for the ICC Manager.

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

Initializes the ICC Manager with all the required default profiles.

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

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

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 CMM handle to the ICC profile contained in the buffer.

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 (display 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.

cmm_profile_t* **gsicc_set_iccsmaskprofile**(const char *pname, int namelen, gsicc_manager_t *icc_manager, gs_memory_t *mem);

Specialized function used in the setting of the soft mask profiles and the gray-to-k profile.

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 c-list 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.

int **gsicc_init_gs_colors**(gs_state *pgs);

Used during start-up to ensure that the initial default color spaces are associated with ICC profiles.

void **gs_setoverrideicc**(gs_imager_state *pis, bool value);

Set the override_internal variable in the icc manager.

bool **gs_currentoverrideicc**(gs_imager_state *pis);

Get the override_internal variable in the icc manager.

void **gs_setoverride_ri**(gs_imager_state *pis, bool value);

Set the override_ri variable in the icc manager.

bool **gs_currentoverride_ri**(gs_imager_state *pis);

Get the override_ri variable in the icc manager.

void **gscms_set_icc_range**(cmm_profile_t **icc_profile);

Set the range values to default of 0 to 1 for the profile data.

void **gsicc_setrange_lab**(cmm_profile_t *profile);

Set the range values to default of 0 to 100 for the first component and -128 to 127 for components two and three.

int **gsicc_set_srcgtag_struct**(gsicc_manager_t *icc_manager, const char* pname,
int namelen);

Initializes the srcgtag_profile member variable of the ICC manager. This is set using -sSourceObjectICC.

```
void gsicc_get_srcprofile(gsicc_colorbuffer_t data_cs, gs_graphics_type_tag_t graphics_type_tag,
                          cmm_srcgtag_profile_t *srcgtag_profile, cmm_profile_t **profile,
                          gsicc_rendering_intents_t *rendering_intent);
```

Given a particular object type this function will return the source profile and rendering intent that should be used it it has been specified using -sSourceObjectICC.

5.2 Device Profile Structure

The device structure contains a member variable called icc_struct, which is of type *cmm_dev_profile_t. The details of this structure are shown below.

```
typedef struct cmm_dev_profile_s {
        cmm_profile_t *device_profile[];      /* Object dependent (and default) device profiles */
        cmm_profile_t *proof_profile;         /* The proof profile */
        cmm_profile_t *link_profile;          /* The device link profile */
        gsicc_rendering_intents_t intent[];   /* Object dependent rendering intents */
        bool devicegraytok;                   /* Force source gray to device black */
        bool usefastcolor;                    /* No color management */
        gs_memory_t *memory;
        rc_header rc;
} cmm_dev_profile_t;
```

There are a number of operators associated with the device profiles. These include:

```
cmm_dev_profile_t* gsicc_new_device_profile_array(gs_memory_t *memory);
```

This allocates the above structure.

```
int gsicc_set_device_profile_intent(gx_device *dev, gsicc_profile_types_t intent,
                                    gsicc_profile_types_t profile_type);
```

This sets the rendering intent for a particular object type.

```
int gsicc_init_device_profile_struct(gx_device * dev, char *profile_name,
```

gsicc_profile_types_t profile_type);

This sets the device profiles. If the device does not have a defined profile, then a default one is selected.

void **gsicc_extract_profile**(gs_graphics_type_tag_t graphics_type_tag,
cmm_dev_profile_t *profile_struct, cmm_profile_t **profile,
gsicc_rendering_intents_t *rendering_intent);

Given a particular object type, this will return the device ICC profile and rendering intent to use.

int **gsicc_set_device_profile**(gx_device * pdev, gs_memory_t * mem, char *file_name,
gsicc_profile_types_t defaulttype);

This sets the specified device profile. This is used by gsicc_init_device_profile_struct, which will specify the default profile to this function if one was not specified.

int **gsicc_get_device_profile_comps**(cmm_dev_profile_t *dev_profile);

Returns the number of device components of the profile associated with the device.

5.3 Link Cache

The Link Cache is a reference counted member variable of Ghostscript's imager state and maintains recently used links that were provided by the CMM. These links are handles or context pointers provided by the CMM and are opaque to Ghostscript. As mentioned above, the link is related to the rendering intents, the object type and the source and destination ICC profile. From these items, a hash code is computed. This hash code is then used to check if the link is already present in the Link Cache. A reference count variable is included in the table entry so that it is possible to determine if any entries can be removed if there is insufficient space in the Link Cache for a new link. The Link Cache is allocated in stable GC memory and is designed with semaphore calls to allow multi-threaded c-list (display list) rendering to share a common cache. Sharing does require that the CMM be thread safe. 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 source and destination buffers when transforming a buffer of color values.

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

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.

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. If the output_colorspace variable is NULL, then the ICC profile associated with dev will be used as the destination color space.

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

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,
gx_device *dev, cmm_profile_t *gs_output_profile,
gsicc_rendering_param_t *rendering_params);

This performs a transformation on the named color given a particular tint value and returns 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.

5.4 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, Windows 7 and Windows 8)), 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_devlink**(gcmmhprofile_t lcms_srchandle,
gcmmhprofile_t lcms_proofhandle,
gcmmhprofile_t lcms_deshandle,
gcmmhprofile_t lcms_devlinkhandle,
gsicc_rendering_param_t *rendering_params);

This function is similar to the above function but includes a proofing ICC profile and/or a device link ICC profile in the calculation of the link transform. See Section 4.2.

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**(gx_device *dev, 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 on chunks of data will occur. 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. Note that a pointer to this function is stored in a member variable of Ghostscript's ICC link structure (gsicc_link_t.procs.map_buffer).

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

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. Note that a pointer to this function is stored in a member variable of Ghostscript's ICC link structure (gsicc_link_t.procs.map_color).

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

This function obtains 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.

int **gscms_get_channel_count**(gcmmhprofile_t profile);

Return the number of colorants or primaries associated with the profile.

int **gscms_get_pcs_channel_count**(gcmmhprofile_t profile);

Get the channel count for the profile connection space. In general this will be three but could be larger for device link profiles.

6 ICC Color, the Display List and Multi-Threaded Rendering

Ghostscript's display list is referred to the c-list (command list). Using the option -dNumRenderingThreads=X, it is possible to have Ghostscript's c-list rendered with X threads. In this case, each thread will simultaneously render different horizontal bands of the page. When a thread completes a band, it will move on to the next one that has not yet been started or completed by another thread. Since color transformations are computationally expensive, it makes sense to perform these operations during the multi-threaded rendering. To achieve this, ICC profiles can be stored in the c-list and the associated color data stored in the c-list in its original source space.

Vector colors are typically passed into the c-list in their destination color space, which is to say that they are already converted through the CMM. Images however are not necessarily pre-converted but are usually put into the c-list in their source color space. In this way, the more time consuming color conversions required for images occurs during the multi-threaded rendering phase of the c-list list. Transparency buffers also require extensive color conversions. These buffers are created during the c-list rendering phase and will thus benefit from having their color conversions occur during the multi-threaded rendering process.

7 PDF and PS CIE color space handling

One feature of Ghostscript is that all color conversions can be handled by the external CMM. This enables more consistent specialized rendering based upon object type and rendering intents. Most CMMs cannot directly handle CIE color spaces defined in PostScript or the CalGray and CalRGB color spaces defined in PDF. Instead most CMMs are limited to handling only ICC-based color conversions. To enable the handling of the non ICC-based color spaces, Ghostscript converts these to equivalent ICC forms. The profiles are created by the functions in gsicc_create.c.

PostScript color spaces can be quite complex, including functional mappings defined by programming procedures. Representing these operations can require a sampling of the 1-D procedures. Sampling of functions can be computationally expensive if the same non-ICC color space is repeatedly encountered. To address this issue, the equivalent ICC profiles are cached and a resource id is used to detect repeated color space settings within the source document when possible. The profiles are stored in the profile cache 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. Instead, a device ICC profile should be used to define the color for the output device. There is currently an enhancement request to enable the option of converting CRDs to equivalent ICC profiles.

Note that gsicc_create.c 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.

8 DeviceN and Separation colors

8.1 Spot Colors

Spot colors, which are sometimes referred to as named colors, are colorants that are different than the standard cyan, magenta, yellow or black colorants. Spot colors are commonly used in the printing of labels or for special corporate logos for example. In PostScript and PDF documents, color spaces associated with spot colors are referred to as separation color spaces. The ICC format defines a structure for managing spot colors called a named color profile. The structure consists of a table of names with associated CIELAB values for 100 percent tint coverage. In addition, the table can contain optional CMYK device values that can be used to print the same color as the spot color. In practice, these profiles are rarely used and instead the proofing of spot colors with CMYK colors is often achieved with proprietary mixing models. The color architecture of Ghostscript enables the specification of a structure that contains the data necessary for these mixing models. When a fill is to be made with a color in a separation color space, a call is made passing along the tint value, the spot color name and a pointer to the structure so that the proprietary function can return the device values to be used for that particular spot color. If the function cannot perform the mapping, then a NULL valued pointer is returned for the device values, in which case the alternate tint transform specified in the PDF or PS content is used to map the spot tint color.

8.2 DeviceN Colors

DeviceN color spaces are defined to be spaces consisting of a spot color combined with one or more additional colorants. A DeviceN color space can be handled in a similar proprietary fashion as spot colors if desired. The details of this implementation are given in Section 8.3. Ghostscript also provides an ICC-based approach for handling DeviceN source colors. In this approach, xCLR ICC source profiles can be provided to Ghostscript upon execution through the command line interface using -sDeviceNProfile. These profiles describe how to map from DeviceN tint values to CIELAB values. The profiles must include the colorantTableTag. This tag is used to indicate the colorant names and the lay-down order of the inks. The colorant names are associated with the colorant names in a DeviceN color space when it is encountered. If a match is found, the xCLR ICC profile will be used to characterize the source DeviceN colors. Note that the colorant orders specified by the names may be different in the source profile, necessitating the use of a permutation of the DeviceN tint values prior to color management. An overview of the process is shown in Figure 7. The directory ./gs/toolbin/color/icc_creator contains a Windows application for creating these DeviceN source ICC profiles. Refer to the README.txt file for details and for an example.

In Microsoft's XPS format, all input DeviceN and 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 assuming an external xCLR ICC profile was not specified as described above.

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 is the default manner by which Ghostscript handles 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 is possible to achieve this by having the source and

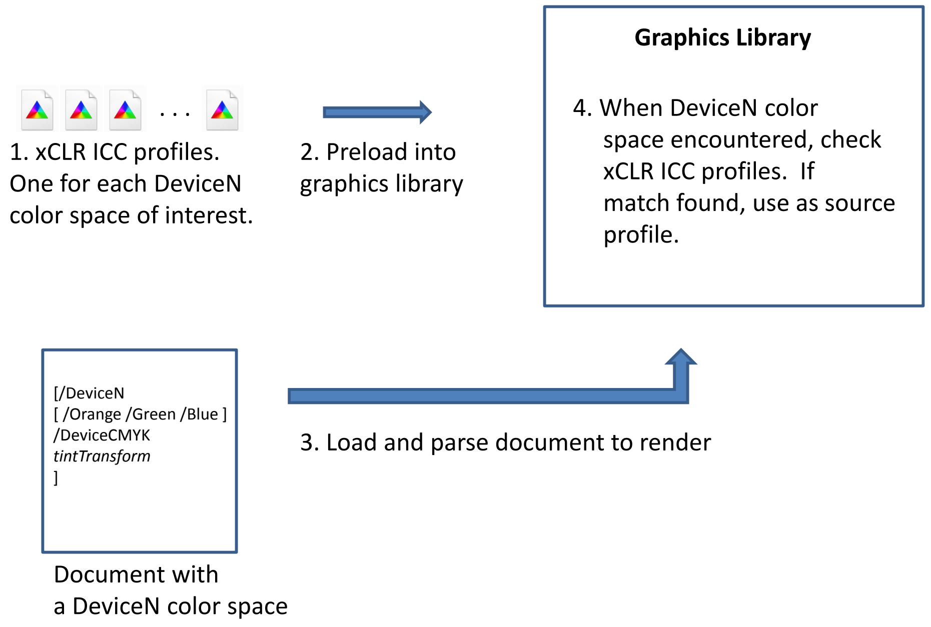


Figure 7: Flow for use of xCLR source profiles to define DeviceN color in PDF and PS source files

destination ICC profiles the same. This will result in an identity transform for 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.

8.3 DeviceN and Spot Color Customization

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 populate the procs of Ghostscript's link structure with the proper pointers to functions that will use the link and transform the colors. The procedure structure, which is a member variable of gsicc_link_t is defined in gscms.h and given as

```
typedef struct gscms_procs_s {
        gscms_trans_buffer_proc_t map_buffer;   /* Use link to map buffer */
        gscms_trans_color_proc_t map_color;     /* Use link to map single color */
        gscms_link_free_proc_t free_link        /* Free link */
} gscms_procs_t;
```

For the CMM that is interfaced with Ghostscript, these procedures are populated with

```
map_buffer = gscms_transform_color_buffer;
map_color = gscms_transform_color;
free_link = gscms_release_link;
```

Assuming the DeviceN color manager can handle the DeviceN color space, when it returns the opaque link handle, which is assigned to the link_handle member variable of gsicc_link_t, the above procs should be populated with the procedures that will actually make use of the link. For example,

```
map_buffer = devn_transform_buffer;
map_color = devn_transform_color;
free_link = devn_release_link;
```

In this case, the graphics library will make the appropriate calls when it is making use of the link. As an example template, The unmanaged color option -dUseFastColor makes use of this approach to provide "links" that use special mapping procedures where

```
map_buffer = gsicc_nocm_transform_color_buffer;
map_color = gsicc_nocm_transform_color;
free_link = gsicc_nocm_freelink;
```

In this way, the fact that unmanaged color is occurring is opaque to Ghostscript. Similarly, the use of special mixing model links for DeviceN color would be unknown to Ghostscript with this approach and requires nothing more than the minor interface procedure settings (as well as the actual code to compute the mixing result).

9 PCL and XPS Support

PCL[5] makes use of the new color management architecture primarily through the output device profiles as source colors are typically specified to be in the sRGB color space.

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

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.