Ghostscript 9.21 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 9.21. 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 1.6

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 9.21 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 source 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), rendering intent and black point compensation 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 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.
- Easy to specify different black point compensation settings for graphic, image and text objects.
- Ability to make use of a PDF output intent ICC profile.
- Ability to use an NCLR ICC output profile when rendering to a separation device.
- Control to force gray source colors to black ink only when rendering to output devices that support black ink.
- Ability to make use of device link ICC profiles for direct mapping of source colors to the device color space.
- Ability to make use of device link ICC profiles for retargeting from SWOP/Fogra standard color space to a specific device color space.
- Ability to monitor for the presence of color on individual pages, which is useful for certain print systems.
- Ability to specify different default transparency blending color spaces.
- Ability to specify a post rendering ICC profile for certain devices.

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), black point compensation setting 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 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

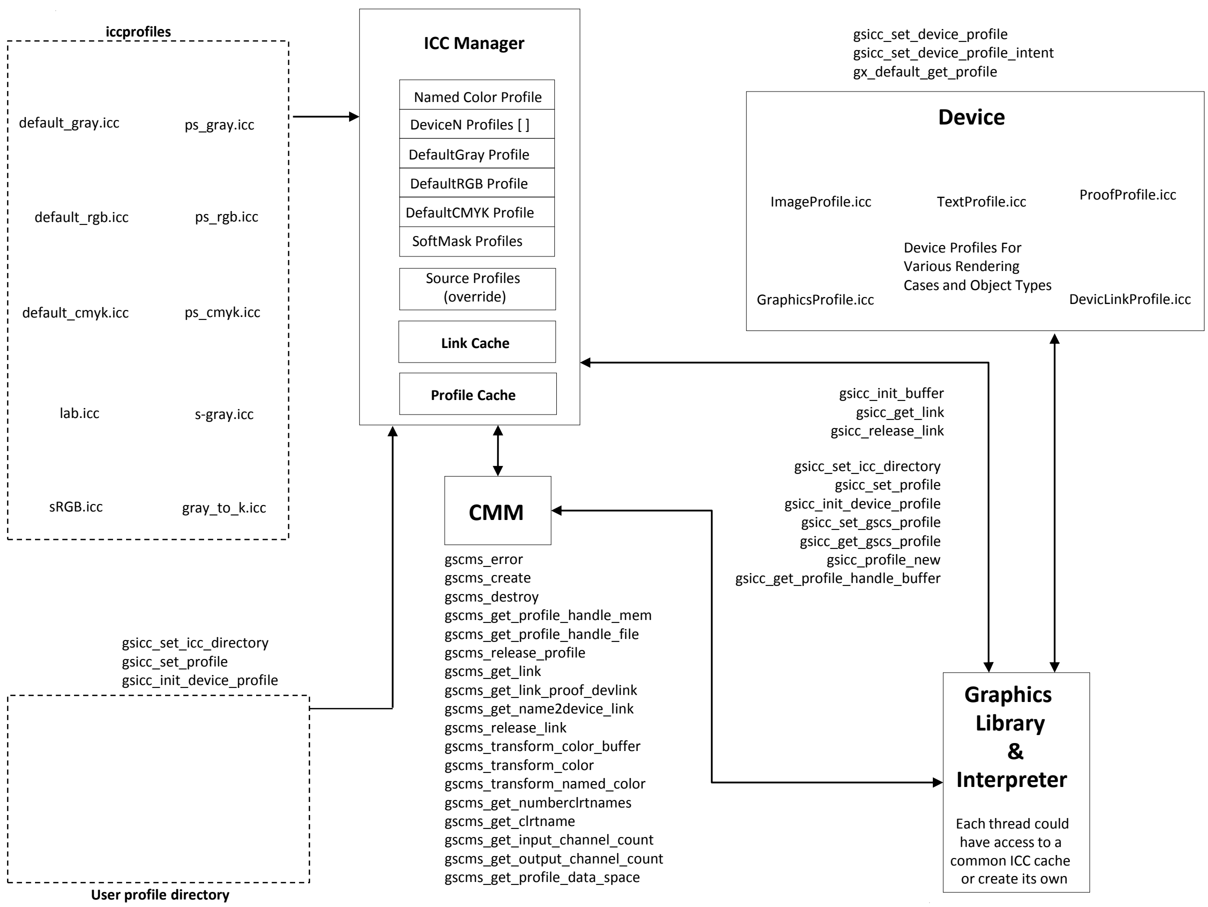


Figure 1: Graphical Overview of Ghostscript's Color Architecture

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:

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

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 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 4.3 for details on this option.

The command line option

where 0 implies compensation is off and 1 implies that compensation if on. Integer values were used instead of boolean for this command to enable easy expansion of the option to different types of black point compensation methods.

It is also possible to make use of the special black preserving controls that exist in littleCMS. The command line option

4.1 Output Intents and Post Rendering Color Management

PDF documents can contain target ICC profiles to which the document is designed to be rendered. These are called output intents within the PDF specification. It is possible to make use of these profiles with the use of the command line option

Note that this allows for the cases where the output intent color space of the document is CMYK based while the output device is RGB based. In such a situation we would use a PostRenderProfile that was RGB based.

4.2 Transparency and Color Management

Transparency blending in PDF can be dependent upon the color space in which the blending takes place. In certain source files, the color space for which the blending is to occur is not specified. Per the specification, when this occurs, the color space of the target device should be used. For consistent output across different device types this is not always desirable. For this reason, Ghostscript provides the capability to specify the desired default blending color space through the command line option link profile. In this case, the profile specified by -sOutputICCProfile would be the profile for the common CMYK space.

Note that if -sSourceObjectICC is used to specify device link ICC profiles to map from source color spaces to device colors, then it is not possible to use either the device profile or the proofing profile for these objects. However, a device link profile that is associated with the target device will be merged with the device link profile specified for the source object.

4.4 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 an unprecedented amount of color control based upon object type.

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

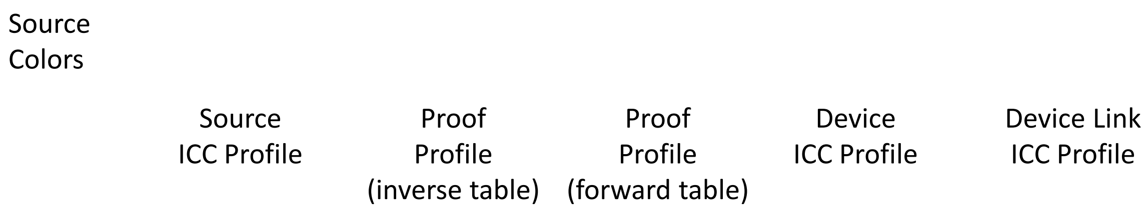


Figure 2: Flow of data through source, proof, destination and device link ICC profiles

specifies the black preserving method that should be used from mapping CMYK to CMYK for text objects. The options are the same as specified for -dKPreserve.

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 GRAY, RGB and CMYK objects. Because this requires the specification of many 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 render in a normal default fashion in this case. Note that it is necessary to include all the possible options in each line. That is, "Graphic_CMYK cmyk_src_cyan.icc 0" is not a valid line but must include settings for the next three values as given above for Graphic_CMYK. In addition to CMYK and RGB types given above, the user can also specify Graphic_GRAY, Image_GRAY and Text_GRAY objects.

In addition, it is possible to have unique color management methods for these object types through two special names which are "None" and "Replace". For example, if our file contained the following two lines

Graphic_CMYK	None
Text_CMYK	Replace

then graphic CMYK source objects will not be color managed but instead will go through the standard Postscript mapping methods (e.g. 255-X). CMYK text objects will go through the color replacement color management route which is provided for those developers who wish to provide direct output replacement colors for a given incoming source color. This is currently implemented in the function gsicc_rcm_transform_general, which is in the file gsicc_replacecm.c. The current implementation computes the color negative of the source color as a demonstration. Note that the replaced color should be in the device's color space. The entire contents of the file, gsicc_replacecm.c are provided as an example for developers.

In addition, one can specify a device link ICC profile to use with a particular source object type when mapping to the destination color space. This is done by simply using a notation such as

Graphic_RGB	linkRGBtoCMYK.icc	0	1	0

in the -sSourceObjectICC file, where linkRGBtoCMYK.icc is the device link ICC profile file name. Note that the rendering intent etc are still specified but their effect is dependent upon the CMM that is hooked in with Ghostscript. With the current use of lcms, these values have no effect with device link profiles. Note also that if the device ICC profile is an NCLR profile, it is possible that the device link profiles specified in the -sSourceObjectICC file can have a destination color space that is either CMYK or NCLR.

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

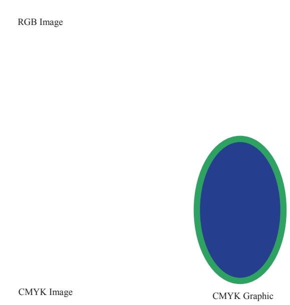


Figure 3: Example file with mixed content. The file includes RGB and CMYK text, graphics, and images

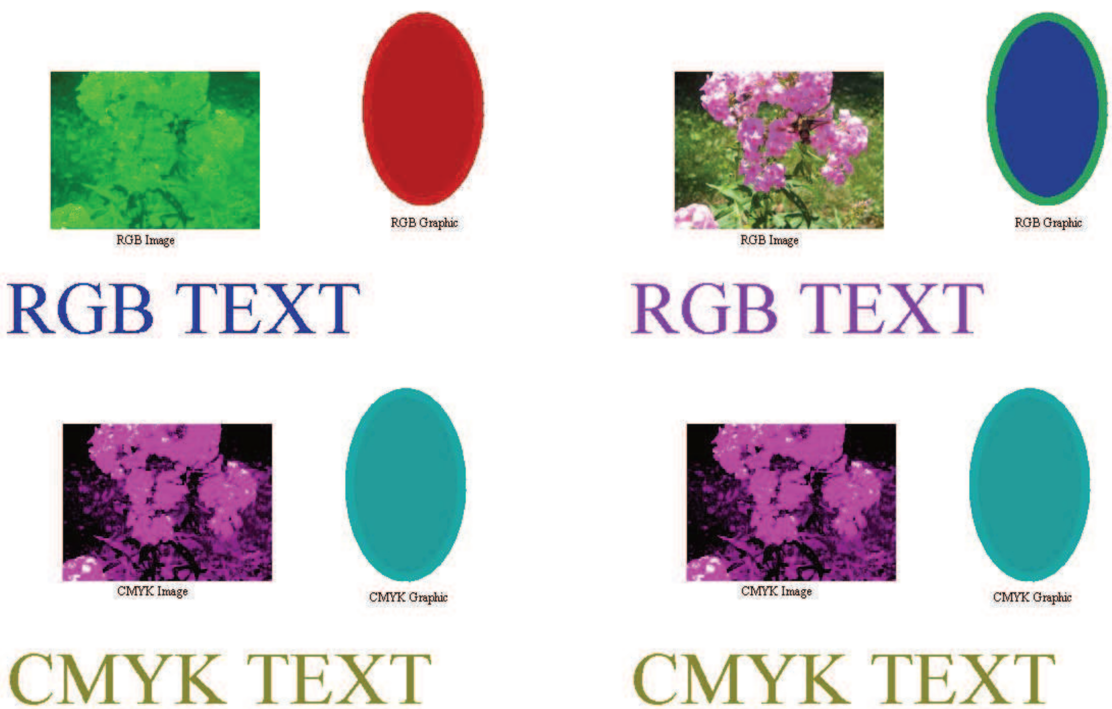
interpretation of "\" by the Microsoft C startup code.

Figure 3 displays the source file text_graph_image_cmyk_rgb.pdf rendered with default settings and Figure 4a 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	1	0	0
Image_CMYK	cmyk_src_renderintent.icc	1	1	0	0
Text_CMYK	cmyk_src_renderintent.icc	2	1	0	0

and rendering the file ./examples/text_graph_image_cmyk_rgb.pdf to an RGB device, one obtains the output shown in Figure 4b. 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



(a) Source profiles vary with object type

(b) Rendering intents vary with CMYK source object type

Figure 4: Examples of object based color transformations for the file from Figure 3 by specifying **source** profiles and/or rendering intents

(a) Destination profiles vary with object type

(b) Destination intents vary with object type

Figure 5: Examples of object based color transformations for the file from Figure 3 by specifying **destination** profiles and/or intents

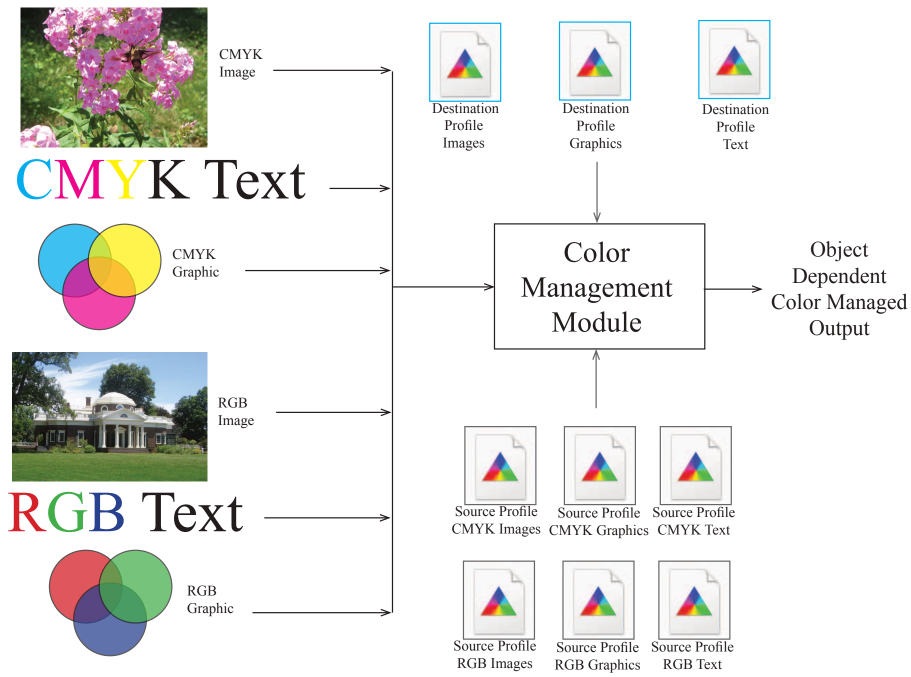


Figure 6: Overview of profiles that can be used in object dependent color management

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 5a 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.

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

Figure 5b 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 6, which shows how both the source and/or the destination ICC profiles can be specified.

5 Details of objects and methods

At this point, let us go into further detail of the architecture and in particular the various functions that may be of interest to those wishing to work with ICC profiles within Ghostscript. 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 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;
```

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

Initializes the ICC Manager with all the required default profiles.

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.

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

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

Several ICC profile-specific operators in gsicc_manage.c/h that may be of interest to developers include the following:

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_clone_profile**(cmm_profile_t *source, cmm_profile_t **destination, gs_memory_t *memory);

Used for cloning an ICC profile. This is used in the multi-threaded rendering case to create thread-safe color management as the threads render to the same device profile.

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

Set the hash code for a profile.

int64_t **gsicc_get_hash**(cmm_profile_t *profile);

Get the hash code for a profile. In gsicc_cache.h/c due to its use in computing links.

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.

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_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_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_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.

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.

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

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

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

Returns the number of device channels for a profile.

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_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 if 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[];
```

This allocates the above structure.

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

This sets a device profile for a particular object type, default type, output intent, post-render, blending color space, proofing or link. 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_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.

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_set_device_blackptcomp**(gx_device *dev, gsicc_blackptcomp_t blackptcomp, gsicc_profile_types_t profile_type);

This sets the black point compensation for a particular object type.

int **gsicc_set_device_blackpreserve**(gx_device *dev, gsicc_blackpreserve_t blackpreserve, gsicc_profile_types_t profile_type);

This sets the black preservation for a particular object type.

void **gsicc_extract_profile**(gs_graphics_type_tag_t graphics_type_tag,
cmm_dev_profile_t *profile_struct, cmm_profile_t **profile,
gsicc_rendering_param_t *render_cond);

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

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. (Defined in gsicc_cache.h/c)

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.

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.

There are special link allocation/free operations that can be invoked that are not tied to the Link Cache. These are typically used in situations where a device may need to create a link for special post rendering color management. The operations are:

gsicc_link_t* **gsicc_alloc_link_dev**(gs_memory_t *memory, cmm_profile_t *src_profile, cmm_profile_t *des_profile, gsicc_rendering_param_t *rendering_params);

This is a special allocation for a link that is used by devices for doing color management on post rendered data. It is not tied into the profile cache like gsicc_alloc_link.

void **gsicc_free_link_dev**(gs_memory_t *memory, gsicc_link_t* *link);

Free link allocated using gsicc_alloc_link_dev.

5.4 Interface of Ghostscript to CMM

Ghostscript interfaces to the CMM through a single file. The file gsicc_littlecms2.c/h is a reference interface between littleCMS and Ghostscript. If a new library is used (for example, if littleCMS is replaced with a different CMM), 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 4.3.

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 as 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. 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. 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.

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.

The use of the command line option -dUseCIEColor will result in document DeviceGray, DeviceRGB and DeviceCMYK source colors being substituted respectively by Postscript CIEA, CIEABC and CIEDEFG color spaces. In this case, -sDefaultGrayProfile, -sDefaultRGBProfile and -sDefaultCMYKProfile will not specify the ICC profiles to use for these source spaces. The PS color spaces that are used with -dUseCIEColor are defined in the directory gs/Resource/ColorSpace within the files DefaultGray, DefaultRGB and DefaultCMYK. Note that Ghostscript will end up converting these PS color spaces to equivalent ICC profiles using the methods in gsicc_create.c, so that the ICC-based CMM can perform the proper color conversions.

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.

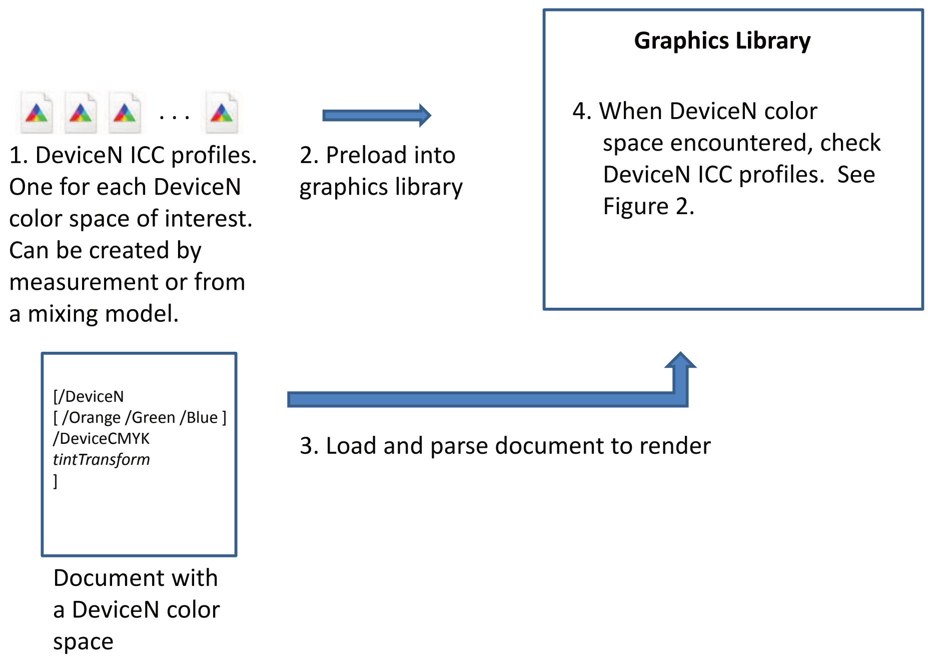


Figure 7: Flow for use of xCLR source profiles to define DeviceN color in PDF and PS source files

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 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, Spot Color Customization and Direct Color Replacement

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 and DeviceN 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

DeviceN color handling can also 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.

In addition to Custom DeviceN and Separation color handling, it is possible to force a path to a completely customizable color management solution for any object type using the

map_buffer = gsicc_rcm_transform_color_buffer;
map_color = gsicc_rcm_transform_color;
free_link = gsicc_rcm_freelink;

This is provided as an example implementation for RIP OEMs desiring to provide unique color solutions for their products. Note that the file gsicc_nocm.c and gs_replacecm.c both use operators tied to the Link Cache to enable the links that are not ICC based to be stored in the same cache as the ICC based ones. If a developer wishes to implement their own color conversion methods and make use of Ghostscript's Link Cache they can do so following the examples in these files.

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.