Ghostscript Color Management

M. J. Vrhel, Ph.D.
Color Scientist
Artifex Software Inc.

Abstract

This document provides information on a redesign of color management within Ghostscript. Due to its long history of development, Ghostscript's color management has been heavily based upon PostScript Color Management. The new design is focused upon an ICC workflow, which is common today in the printing community. The new design provides significant flexibility for customization by 3rd parties including the ability to interface to other color management modules. This document provides an overview of the architecture as well as usage and developer information.

Introduction

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

The overall motivation for this work is to modernize the color flow of Ghostscript and in particular make it much easier for our customers to use their own color management systems (CMS). Many RIP manufacturers have designed their own color management systems to provide a marketing advantage over their competitors

Today, almost all print color management is performed using ICC profiles as opposed to PostScript Color Management (PCM), which predates the ICC format. 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. The current methods used for performing device independent color in Ghostscript are computationally costly due to the use of multiple transforms. This new color flow addresses that issue through the use of linked transforms that can be readily applied to buffers of data.

Many customers wish to apply different color transformations depending upon if the object being drawn is an image, a graphic or text. The common example is that it is desirable to use pure black as opposed to composite black when drawing black text. In addition, with an image, you likely will want to use a more perceptual based gamut mapping method but with a graphic you likely want to use a more saturated method.

Given these many concerns, the requirements of the architecture are as follows:

- It must be easy to introduce a new CMS into the build of Ghostscript.
- There must be objects/methods to manage the ICC profiles and the linked transforms.
- It must be possible to define all color spaces in terms of ICC profiles.
- It must be possible to have the CMS operate on buffers of data.
- Devices can communicate ICC profiles and have their ICC profile set.
- It must work with PostScript color management definitions.
- It must be possible to cache the linked transformations which define the mapping from one color space to another so that we avoid having to recompute links when we have frequently changing color spaces. In addition, it will be necessary to do lazy linking of these mappings (e.g. only create a mapping when requested to transform data)

- It must have the ability to incorporate the object type (e.g. image, graphic, text) and rendering intent into the computation of the linked transform.
- It must be designed to operate efficiently in a multithreaded environment.

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 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, and finally the *color management system* (CMS), which is the external engine that provides and performs the linked transformations (e.g. littleCMS).

In the typical flow, when a thread is ready to transform a buffer of data, it will request a linked transform from the Link Cache. When requesting a link, it is necessary to provide information to the CMS, 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 CMS 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 CMS, it is possible that the CMS 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 directly to the CMS. 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.

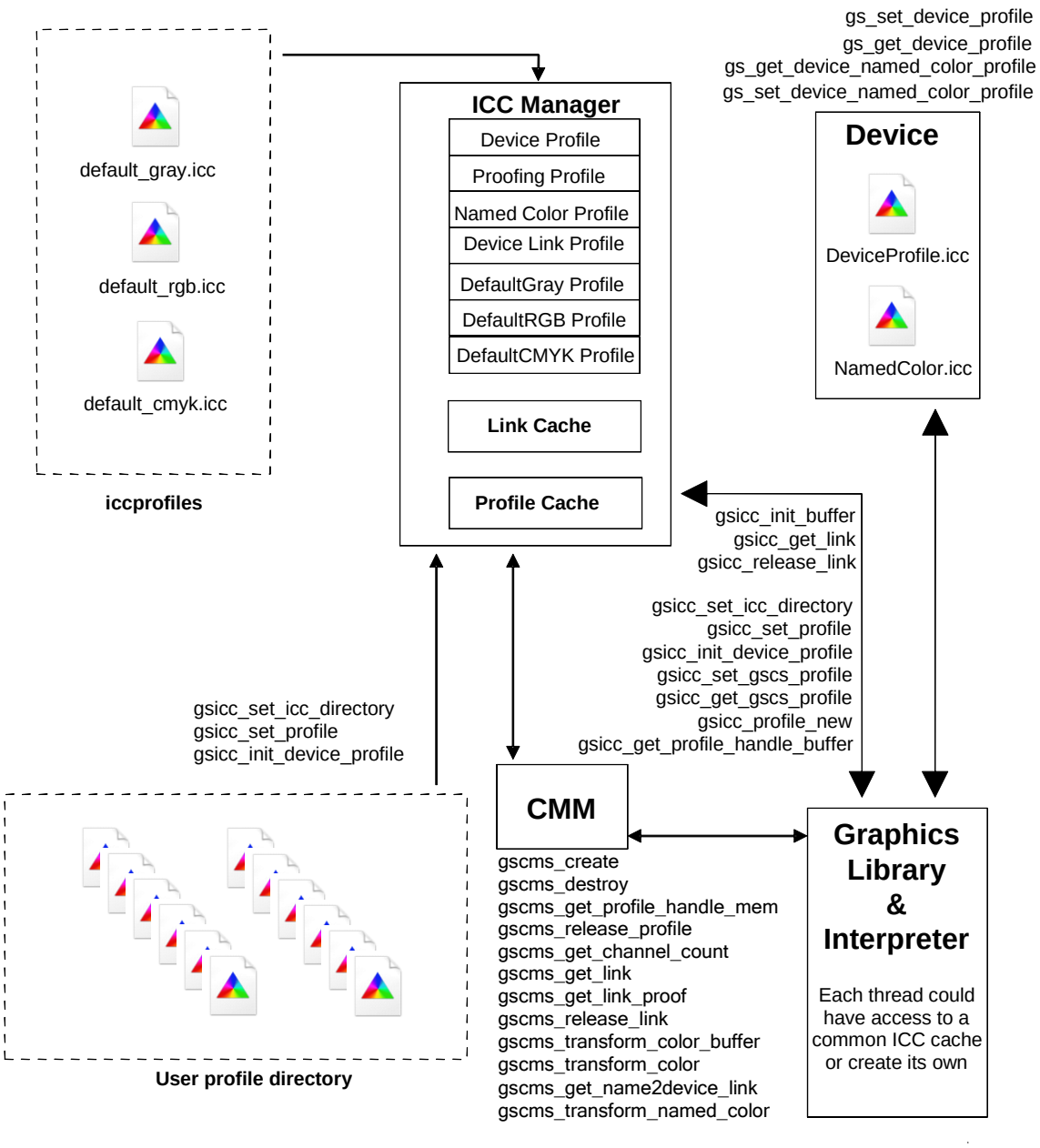

Figure 1: Graphical overview of ICC architecture.

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 and PostScript (PS) have a distinction between process colors and device colors. Figures 3-10 provide an overview of how color is defined in various PDLs. As seen in Figures 3 and 5, there is a conversion (e.g. via UCR/BG) from device colors to process colors for PDF and PS. 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 an 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.

The remaining steps shown in Figures 3 and 5 consist of transfer functions and halftone functions. It is possible to pack the transfer functions into the ICC profile or have them externally defined as part of the Postscript or PDF file. It is up to the user to handle this in their desired manner (i.e. they need to design their device ICC profile appropriately). Halftoning of course occurs after color conversion as shown in Figure 3 and 5.

Usage

The code is currently available for checkout with SVN on a branch named icc_work of the main trunk of Ghostscript. The specific URL is

http://svn.ghostscript.com/ghostscript/branches/icc_work

The branch is built in the same process by which the trunk of Ghostscript is built. See the documentation at www.ghostscript.com for details.

The ICC branch introduces several new command line options that can be invoked for complete color management control.

To define source colors that are not already colorimetrically defined in the source document, the following command line options can be invoked.

-sDefaultGrayProfile=my_gray_profile.icc
-sDefaultRGBProfile=my_rgb_profile.icc
-sDefaultCMYKProfile=my_cmyk_profile.icc

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

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

-sOutputICCProfile=my_device_profile.icc

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/

Warnings will be emitted when running a debug version if problems occur with respect to finding the ICC profiles and it is possible that the program may terminate.

There are additional optional settings that are currently under development. These include

-sProofProfile=my_proof_profile.icc
-sNamedProfile=my_namedcolor_profile.icc
-sDeviceLinkProfile=my_link_profile.icc

Setting a proofing profile will make the color management system link multiple profiles together to emulate the device defined by the proofing profile.

If a named color profile is set, then when named colors are encountered in the document they will be mapped to the proper device values. Note that the code does not require that

an ICC profile be used for the named color profile. This is all customizable in the interface code to the CMS. See the details regarding gscms_transform_named_color later in the document

Finally, it will be possible to include a device link profile for other color work flows. For example, this may be useful for devices that output raster content in a standard color space such as SWOP or Fogra CMYK, but they wish to redirect this output to other CMYK devices. While it is possible to handle such a flow in other manners (e.g. using a proofing profile) this is a workflow that is not uncommon.

Finally, note that command line options for XPS and PCL are currently under design.

Overview of objects and methods

At this point, let us go into further detail of the architecture.

ICC Manager

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

- Store the required profile information to use for gray, RGB, and CMYK source colors that are NOT colorimetrically defined in the source document. These entries must always be set in the manager and are set to default values unless defined by the command line interface.
- Store the required profile information for the output device.
- Store the optional profile information related to named colors (if set), the proofing profile (if set) a final output link profile (if set).
- Store the directory be used to search for ICC profiles specified for the above objects.

The manager is created when the imaging state object is created for the graphics library. It is reference counted and allocated in garbage collected memory that is not stable with graphic state restores.

The default gray, RGB and CMYK ICC color spaces as well as the device ICC color space are defined immediately during the initialization of the graphics library. If no ICC profiles are specified externally, then the ICC profiles that are contained in the root folder iccprofiles will be used.

The ICC Manager is defined by the structure given below.

```
typedef struct gsicc_manager_s {
```

```
    cmm_profile_t *device_named;    /* The named color profile for the device */
    cmm_profile_t *default_gray;    /* Default gray profile for device gray */
    cmm_profile_t *default_rgb;     /* Default RGB profile for device RGB */
    cmm_profile_t *default_cmyk;    /* Default CMYK profile for device CMKY */
    cmm_profile_t *proof_profile;   /* Profiling profile */
    cmm_profile_t *output_link;     /* Output device Link profile */
    cmm_profile_t *device_profile;  /* The actual profile for the device */

    char *profiledir;               /* Directory used in searching for ICC profiles */
    uint namelen;

    gs_memory_t *memory;
    rc_header rc;

} gsicc_manager_t;
```

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

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

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

int **gsicc_set_profile**(const gs_imager_state * pis, const char *pname, int namelen, gsicc_profile_t defaulttype);

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

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

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

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

> Creator for the ICC Manager.

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

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

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

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

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

Returns the cmm_icc_profile_data member variable of the gs_color_space object.

gcmmhprofile_t **gsicc_get_profile_handle_buffer**(unsigned char *buffer);

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

Link Cache

The Link Cache is a reference counted member variable to Ghostscript's imager state. Its function is to maintain a list of recently used links that had been provided by the CMS. Currently the cache is simply a linked list where each link has hash information that defines the link in terms of the source ICC profile, the destination ICC profile, and the rendering parameters. The Link Cache is allocated in stable GC memory.

Operators that relate to the Link Cache are contained in the file gsicccache.c/h and include the following:

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

Creator for the Link Cache.

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

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

gsicc_link_t* **gsicc_get_link**(gs_imager_state * pis, gs_color_space *input_colorspace, gs_color_space *output_colorspace, gsicc_rendering_param_t *rendering_params, gs_memory_t

*memory, bool include_softproof);

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

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

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

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 CMS.

CMS

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

Specifically, the functions are as follows:

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

This operation performs any initializations required for the CMS.

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

This operation performs any cleanup required for the CMS.

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 CMS release the profile handles it has created.

int **gscms_get_channel_count**(gcmmhprofile_t profile);

Provides the number of colorants associated with the ICC profile.

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

This is the function that obtains the linkhandle from the CMS. 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 may be useful in the case of the XPS interpreter, where minor color management needs to occur to properly handle gradient stops.

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

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

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

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

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

This is the function through which all color transformations will occur if they are to go through the CMS. This function will be called in the code anytime that we are transforming color from the current graphic state color to the Output Device

color space or to the Blending Color Space, or out of the Blending color space to the Color Space of the parent layer in the transparency stack. Note that if the source hash code and the destination hash code are the same, the transformation will not occur as the source and destination color spaces are identical. This feature can be used to enable "device colors" to pass unmolested through the color processing.

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

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

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

Get a device value for the named color. Since there exist named color ICC profiles and littleCMS supports them, the code in gsicc_littlecms.c is designed to use that format. However, it should be noted that this object need not be an ICC named color profile but can be a proprietary type table. Some CMMs do not support named color profiles. In that case, or if the named color is not found, the caller should use an alternate tint transform or other method. If a proprietary format (nonICC) is being used to define named colors, this operator and **gscms_get_name2device_link** given below must be implemented with that particular format. Note that we allow the passage of a tint value also. Currently the ICC named color profile does not provide tint related information, only a value for 100% coverage. It is provided here for use in proprietary methods, which may be able to provide the desired effect. In gsicc_littlecms.c, a direct tint operation will be applied to the returned device value.

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. Again, the file gsicc_littlecms.c is designed to use ICC named color profiles. Other formats can be easily implemented.

PDF and PS CIE color space handling

If a color space is a PDF or PostScript (PS) CIE color space type (other than ICC), these color spaces will be converted to appropriate ICC objects. The hash code associated with these objects will be based upon the PS or PDF objects as opposed to the created ICC data. Procedural sampling will be performed for the procedures found in PS. Since the blending color spaces are limited to ICC, device or CIE color spaces defined in PDF, the transformation of all to an ICC type is straight forward. The conversion from these spaces to ICC forms is contained in the file gsicc_create.c. Since this file is only needed by the PS and PDF interpreter, it is contained in the psi subdirectory of Ghostscript's folder tree and is not needed for PCL or XPS builds. Performing this conversion, enables the ICC based CMS full control over all color management. To avoid frequent conversions due to frequent color space changes, these color spaces will be cached and indexed related to their resource IDs. This is the profile cache item that is indicated in Figure 1.

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

Device Interaction

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

DeviceN Color Spaces

In obtaining a link transform, if an input color space is DeviceN or Separation type, the XPS is required to have an associated ICC profile. This is not the case with PDF. PDF would have its tint transform and alternate space, which could be any of the CIE or device color spaces which, are handled above. This would be the processing path for handling PDF DeviceN colors if the output device does not understand the colorants defined in the DeviceN color space.

For cases when the device does understand the spot colorants, the preferred handling of DeviceN varies. Many prefer to color manage the CMYK components with a defined CMYK profile, while the other spot colorants pass through unmolested. This will be the default manner by which Ghostscript will handle DeviceN input colors. In other words, if the device profile is set to a particular CMYK profile, and the output device is a separation device, which can handle all spot colors, then the CMYK process colorants will be color managed, but the other colorants will not be managed.

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.

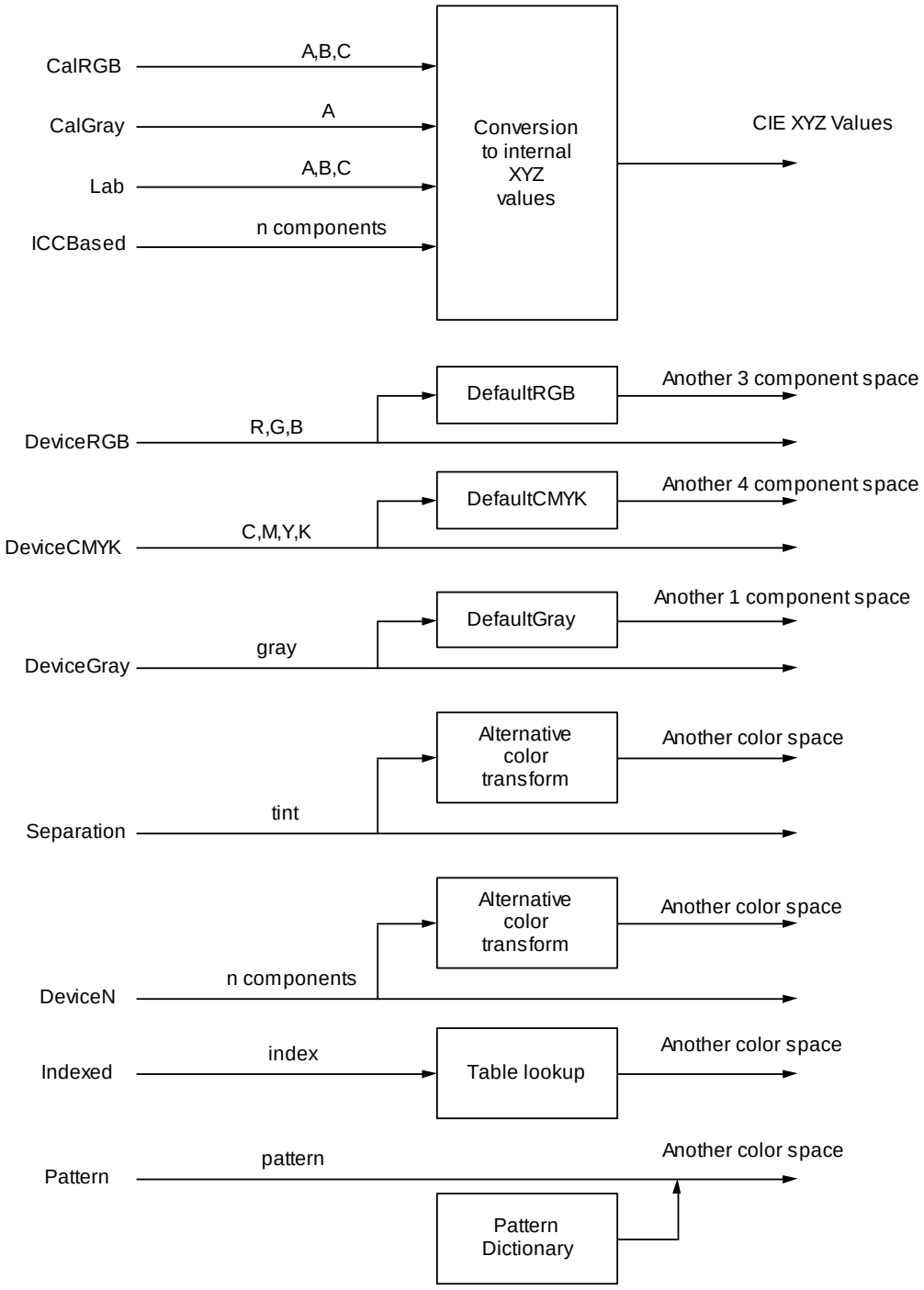

Figure 2: PDF Color Specification

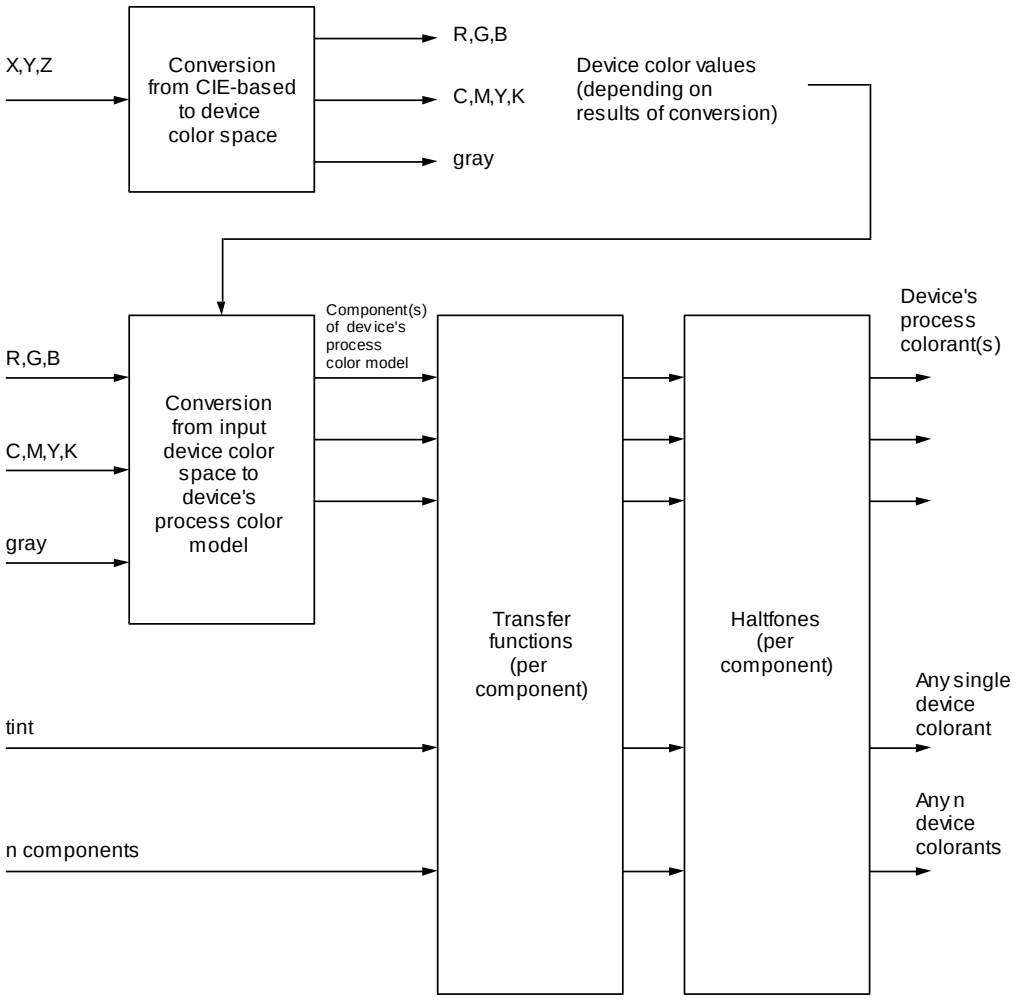

Figure 3: PDF Color Rendering

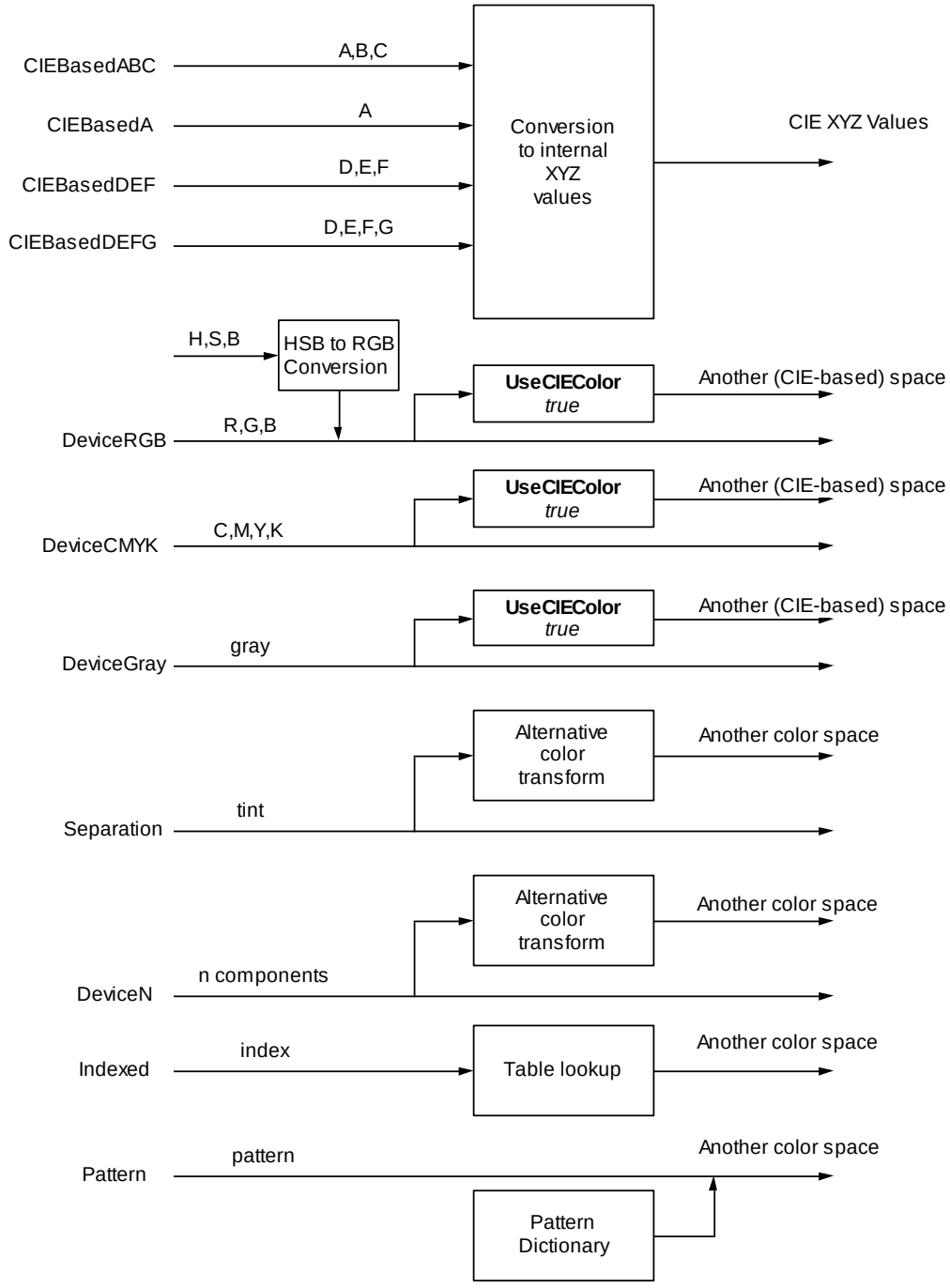

Figure 4: PostScript Color Specification

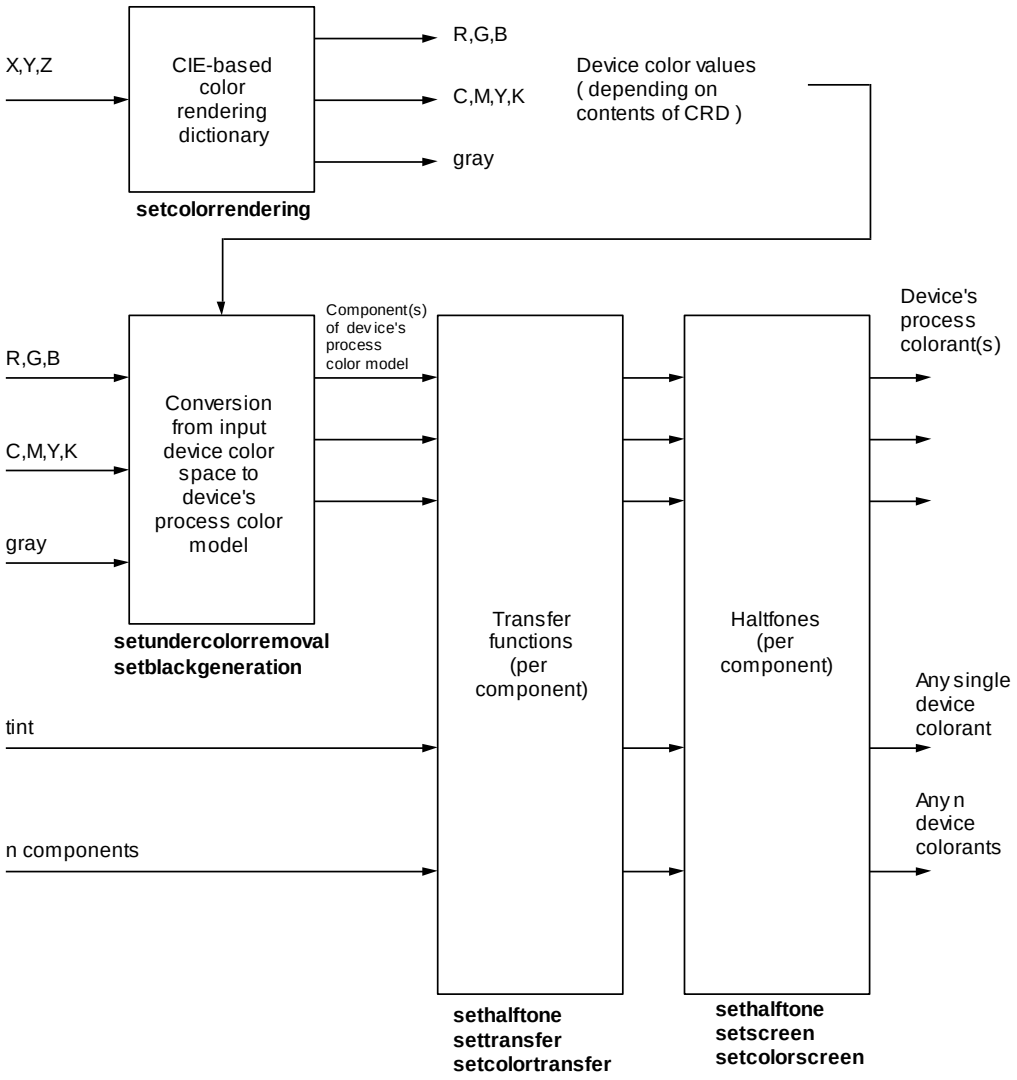

Figure 5: PostScript Color Rendering

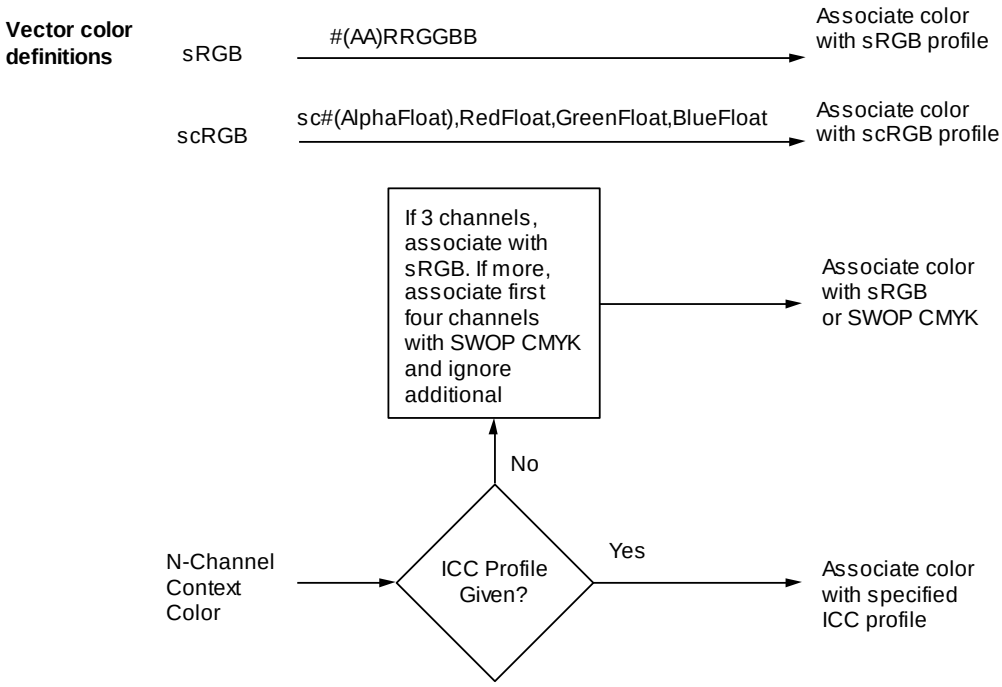

Figure 6: XPS Vector Color Specification

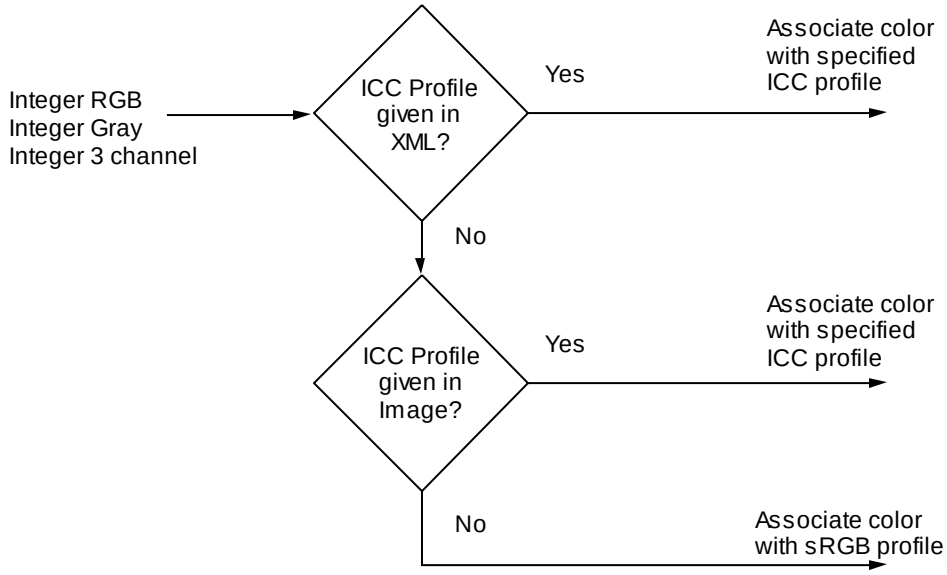

Figure 7: XPS Integer RGB and Grayscale Image Color Specification

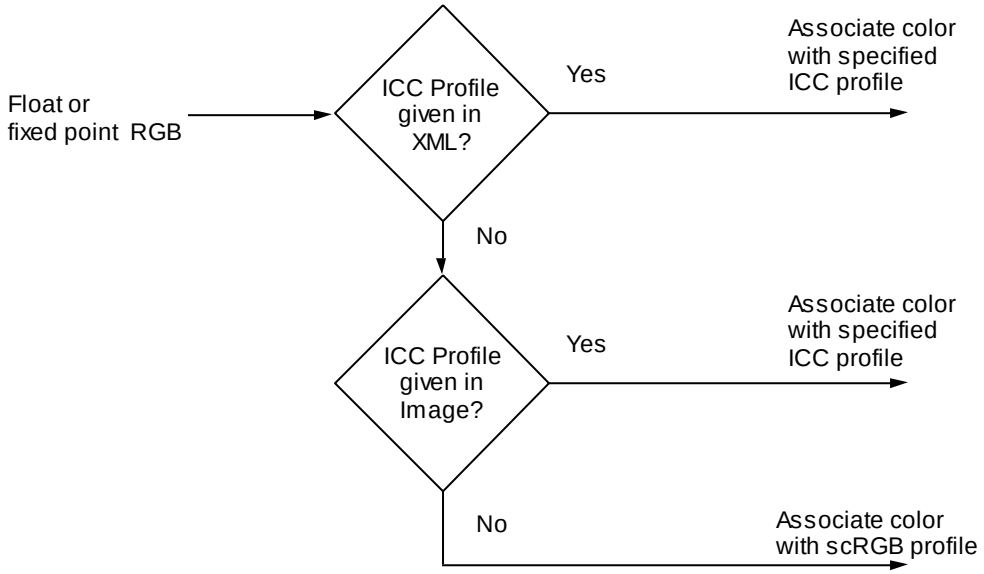

Figure 8: XPS Float or fixedpoint RGB Image Color Specification

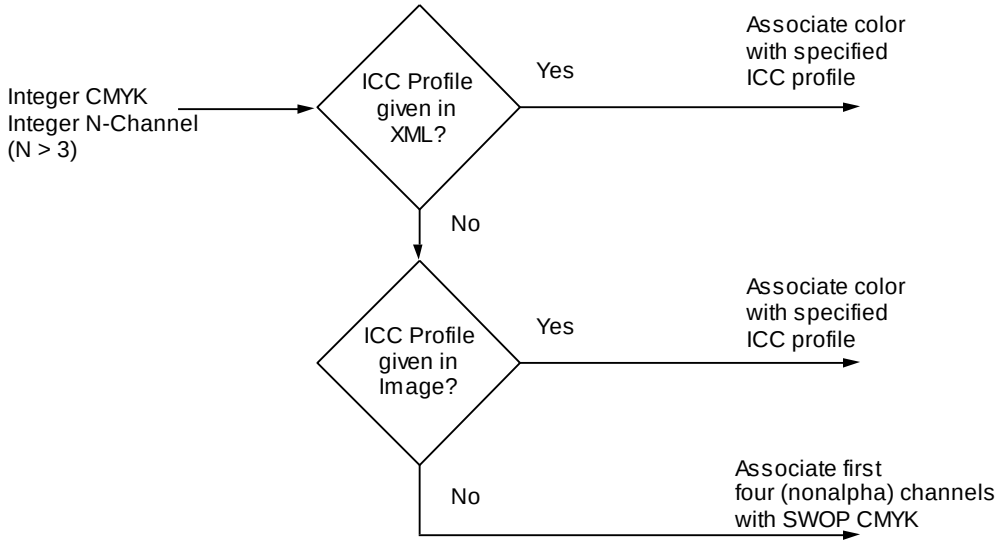

Figure 9: XPS CMYK and N-Channel (N>3) Image Color Specification

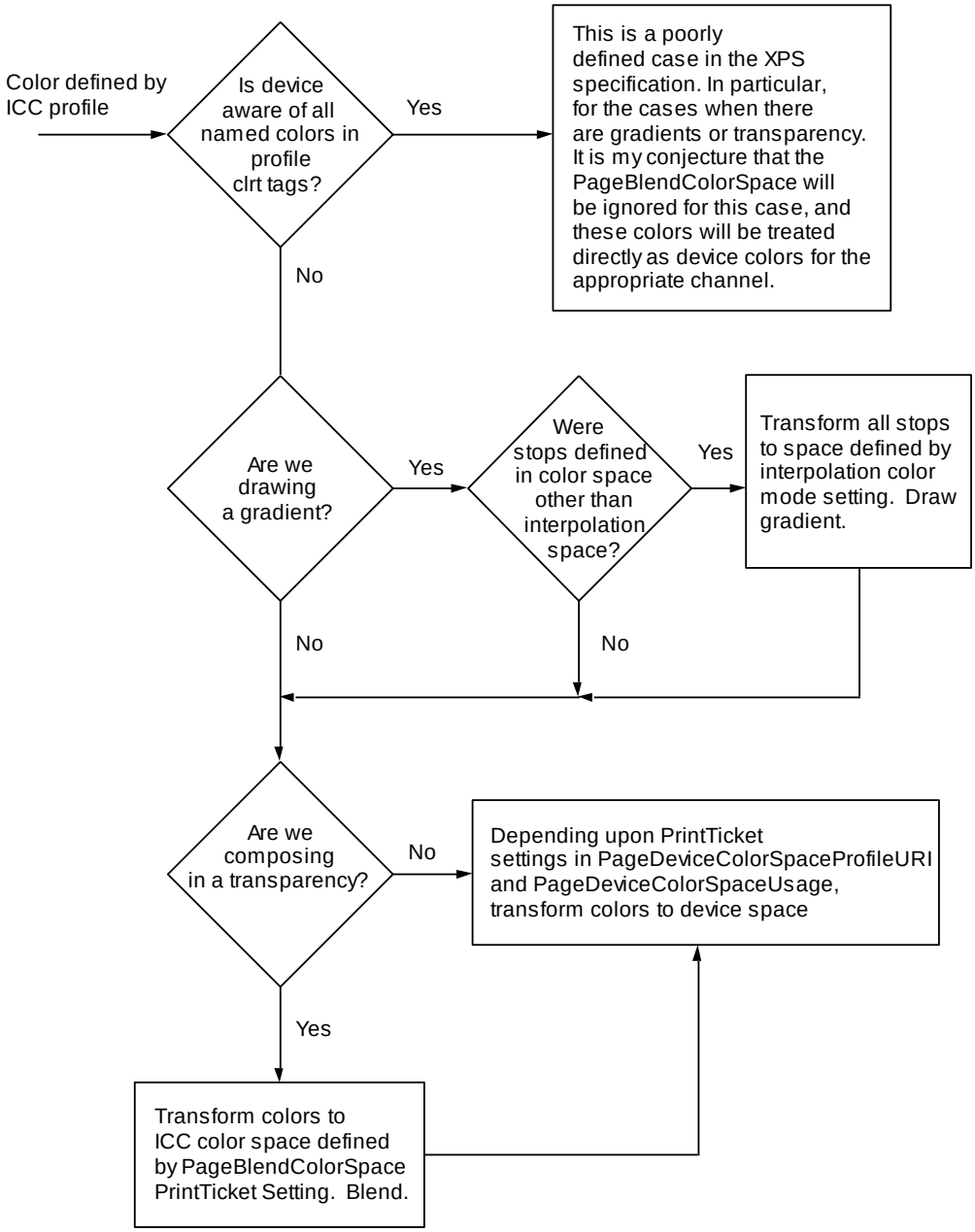

Figure 10: XPS Rendering Side. Note that ALL colors are defined by an ICC color space at this point.