Color Separation Conventions for PostScript Language Programs

Adobe Developer Support

Technical Note #5044

24 May 1996

Adobe Systems Incorporated

Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

PN LPS5044

Contents

Color Separation Conventions for PostScript Language Programs

1 Introduction

This document replaces technical note #5044, “Proposal for Color Separation Conventions for PostScript Language Programs,” last modified on December 14, 1989. The aforementioned document proposed various commenting and coding conventions for PostScript language files for enabling post-processing utilities to create color separations. Many of those conventions were adopted by the industry, a few were not. This paper documents current practice in regards to which conventions have been adopted by standard prepress and page-layout applications, such as Adobe PageMaker, QuarkXPress®, Adobe PrePrint™ Pro, and Adobe TrapWise™. Other enhancements to the paper include the addition of some introductory material on Level 2 in-RIP separations, and a number of examples of composite PostScript language output.

The intended audience for this paper is developers of PostScript drivers and applications that generate their own PostScript language output. This document explains how PostScript language code should be structured so that it will color separate properly in other programs. The paper gives a high-level overview of how to create color separations from composite PostScript language files; however, it does not provide any rigorous sample code for this purpose. Although this document is directed toward authors of composite output, developers of applications that perform color separation should be familiar with its contents, and should make special note of the specific recommendations offered in Appendix B.

The reader should have a knowledge of prepress printing processes and terminology. For those not familiar with prepress nomenclature, Appendix C lists a number of reference documents.

Section 2 of this document explains how to achieve color separations in both PostScript language Level 1 and Level 2 environments. Section 3 introduces the color separation convention operators and comments. Section 4 describes additional guidelines and restrictions for composite PostScript language files. Section 5 concludes the document with several code samples which conform to the color separation conventions.

2 Creating Separations from Composite Documents

How color separations are achieved will depend on whether the PostScript language file is separated by a utility on the host computer or by the printer's raster image processor (RIP). When printing to Level 2 PostScript output devices that are capable of in-RIP separations, color separations are produced by adding a few lines of code to the top of a composite PostScript language job and sending it to the printer. When printing to Level 1 devices, and Level 2 devices that don't have this in-RIP separation feature, creating separations is more difficult, and time consuming, and imposes added constraints on the composite Postscript language code. This latter method of creating separations is called "Level 1-style separations".

An understanding of color separation methods is necessary to appreciate the role that the color separation conventions play in this process. This section will give an overview of both Level 2 in-RIP separations and Level 1-style separations.

2.1 Level 2 in-RIP Separations

Since Level 2 in-RIP separations are the most straightforward, we will first demonstrate how these are achieved, then we will discuss Level 1-style separations.

Making color separations in the RIP entails adding a few lines of set-up code to the composite PostScript language file; the code must do the following:

1. Set the page device Separations key to true.

2. Set the page device ProcessColorModel key to /DeviceCMYK.

3. (optional) Set the page device SeparationColorNames array to the list of all process or custom color inks that your document contains. The names of the colorants of the native color space are included implicitly, regardless of the contents of the array; thus, the empty array [ ] is equivalent to [/Cyan /Magenta /Yellow /Black], after you have performed step 2, above.

4. (optional) Specify which separations to produce and the order that the separations should be output in, using the SeparationOrder page device key. Legal values are the names of the colorants of the native color space, as well as any additional names in the SeparationColorNames array. An empty array [ ] requests that separations for all colors of the native color space, as well as all colors in the SeparationColorNames array, be produced in an unspecified order. In our example below, we request that only the cyan and black process color separations be produced, in that order.

The setpagedevice keys referenced above are described in more detail in the "PostScript Language Reference Manual Supplement for version 2016".

Example 1 below describes a page containing two process colors: cyan and black. The composite page description is sent only once and the output device produces two separations from this data, as shown in Figure 1. If a file consists of more process colors and additional custom colors, the composite representation of the job still only needs to be sent once for the entire color separation job.

Example 1: *Level 2 in-RIP separation example*

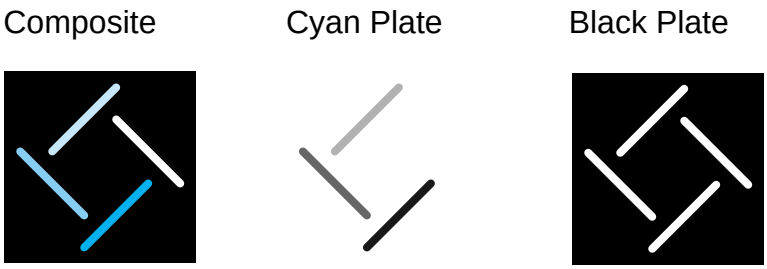
```
%!
%%BoundingBox: 72 72 144 144
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%DocumentProcessColors: Cyan Black
%%EndComments
%%BeginProlog
%%EndProlog
%%BeginSetup
%%EndSetup

%%Page: 1 1
%%BeginPageSetup
% The following setpagedevice call accomplishes steps 1, 2 and 4
% Step 3 is unnecessary for our example.
<<
  /Separations true
  /ProcessColorModel /DeviceCMYK
  /SeparationOrder [/Cyan /Black]
 >> setpagedevice
%%EndPageSetup
%%BeginDocument: (testfiles/file1)
%!PS-Adobe-3.0
%%BoundingBox: 0 0 72 72
%%DocumentProcessColors:  Cyan Black
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%EndComments
%%BeginProlog
%%EndProlog
%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup
% The remainder of the code describes a black box with 4 rotated
% lines on it. The lines are painted with shades of cyan, ranging
% from 0% to 90% ink coverage, as shown in Figure 1.

100 100 translate
gsave
    72 120 div dup scale
    60 60 translate
    newpath -60 -60 moveto 120 0 rlineto
    0 120 rlineto -120 0 rlineto closepath
    0 setgray gsave fill grestore
    1 setlinejoin 1 setlinecap
    0 1 3 { % for
```

```
        0.3 mul 0 0 0 setcmykcolor
        5 setlinewidth
        newpath 50 -10 moveto
        -40 40 rlineto stroke
        90 rotate
    } for
grestore
%%PageTrailer
pgsave restore showpage
%%Trailer
%%EOF
%%EndDocument
%%Trailer
%%EOF
```

Figure 1 *Output from Example 1*



2.2 Level 1-Style Separations

Fundamentally, PostScript language Level 1-style color separation works by redefining some operators and procedures to change their behavior. The file to be separated must be executed with these redefinitions.

As a simplistic example, suppose you want to print the cyan separation of a document. One approach to this is to redefine the **setcmykcolor** operator, which specifies tints of cyan, magenta, yellow, and black ink. If you “throw away” the magenta, yellow, and black, you have the percentage of pure cyan ink. For a color separation, you are printing in black and white; the black areas will eventually have colored ink applied to them. Therefore, what you want as an end result is to print a percentage of black that corresponds to the percentage of cyan that was specified:

```
/setcmykcolor { % def
    pop pop pop 1 exch sub setgray
} bind def
```

This is a simplistic example of what a color separation program may do to output the cyan color separation. To complicate the scenario slightly, you may have a black shape that is partially covered by a cyan shape. In order for

the cyan shape to print as pure cyan, you will not want any of the black ink to print where the two shapes overlap. The following example addresses that problem, by having the cyan shapes print as white on the black plate.

In the example, we use a Level 1-style separation method to color separate the same composite file as in Example 1. To produce two process color separations, the original composite job must be sent twice. First, we emit a few lines of set-up code to print the cyan separation, followed by the composite job, then we emit set-up code for the black separation, followed by the composite job. If a document contains all four process colors, the original composite job must be sent to the printer four times. For complicated jobs, this separation technique may require sending significantly more data to the printer than in the in-RIP separation case, thus resulting in a much longer transmission and processing time.

In Example 2, the **setcmykcolor** and **setgray** operators are redefined for the cyan plate so that tints of cyan print in corresponding tints of black and all other colors print as white (no ink). For the black plate, the **setcmykcolor** operator is redefined so that black prints as various tints of black and other colors print as white (no ink).

Example 2: *Level 1-style separation example*

```
%!
%%BoundingBox: 72 72 144 144
%%DocumentProcessColors:  Cyan Black
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%Pages: 2
%%EndComments
%%BeginProlog
%%EndProlog
%%BeginSetup
%%EndSetup

%%PlateColor: Cyan
%%Page: 1 1
%%BeginPageSetup
save
4 dict begin
   /HalftoneType 1 def
   /Frequency 133 def
   /Angle 15 def
   /SpotFunction { abs exch abs 2 copy add 1 gt {
      1 sub dup mul exch 1 sub dup mul add 1 sub
   }{
      dup mul exch dup mul add 1 exch sub
   } ifelse } bind def
currentdict end
sethalftone

% redefine setcmykcolor and setgray
/oldsetgray /setgray load def
/setcmykcolor {pop pop pop 1 exch sub oldsetgray} bind def
```

```
/setgray {pop 1 oldsetgray} def
%%EndPageSetup

%%BeginDocument: (testfiles/file1)
%!PS-Adobe-3.0
%%BoundingBox: 0 0 72 72
%%DocumentProcessColors:  Cyan Black
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%EndComments
%%BeginProlog
%%EndProlog
%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup
100 100 translate
gsave
    72 120 div dup scale
    60 60 translate
    newpath -60 -60 moveto 120 0 rlineto
    0 120 rlineto -120 0 rlineto closepath
    0 setgray gsave fill grestore
    1 setlinejoin 1 setlinecap
    0 1 3 { % for
        0.3 mul 0 0 0 setcmykcolor
        5 setlinewidth
        newpath 50 -10 moveto
        -40 40 rlineto stroke
        90 rotate
    } for
grestore
%%PageTrailer
pgsave restore showpage
%%Trailer
%%EOF
%%EndDocument
%%PageTrailer
restore

%%PlateColor: Black
%%Page: 1 2
%%BeginPageSetup
save
4 dict begin
   /HalftoneType 1 def
   /Frequency 133 def
   /Angle 45 def
   /SpotFunction { abs exch abs 2 copy add 1 gt {
      1 sub dup mul exch 1 sub dup mul add 1 sub
   }{
      dup mul exch dup mul add 1 exch sub
   } ifelse } bind def
currentdict end
sethalftone

% redefine setcmykcolor
/setcmykcolor {4 1 roll pop pop pop 1 exch sub setgray} bind def
```

```
%%EndPageSetup
%%BeginDocument: (testfiles/file1)
%!PS-Adobe-3.0
%%BoundingBox: 0 0 72 72
%%DocumentProcessColors:  Cyan Black
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%EndComments
%%BeginProlog
%%EndProlog
%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup
100 100 translate
gsave
    72 120 div dup scale
    60 60 translate
    newpath -60 -60 moveto 120 0 rlineto
    0 120 rlineto -120 0 rlineto closepath
    0 setgray gsave fill grestore
    1 setlinejoin 1 setlinecap
    0 1 3 { % for
        0.3 mul 0 0 0 setcmykcolor
        5 setlinewidth
        newpath 50 -10 moveto
        -40 40 rlineto stroke
        90 rotate
    } for
grestore
%%PageTrailer
pgsave restore showpage
%%Trailer
%%EOF
%%EndDocument
%%PageTrailer
restore

%%Trailer
%%EOF
```

2.3 Supporting Overprinting, Images and Spot Colors

Redefining Level 1 operators to produce separations will work with some basic types of documents, as shown above. However, certain types of documents cannot be described using Level 1 PostScript operators, in a manner that allows easy host-based separation. For example, documents that contain:

1. custom (spot) colors, in addition to process colors;

2. objects that have been trapped by overprinting inks in specific areas of the document; and

3. special types of images, such as duotones or tritones.

The color separation conventions specify how to structure PostScript language documents which include images, spot colors, and overprinted inks, so that they will separate properly, even when Level 1-style separations are used.

If your users wish to create output for use with in-RIP separations, you do not need to impose any restrictions on your application's PostScript language code; however, your output should include any relevant DSC color comments as described in section 3.3 of this document. If, on the other hand, your users choose to create output for export to a page layout or prepress program to perform host-based separations, then your output should conform to the color separation conventions described in the remainder of this document.

3 Color Related Operators and Comments

To conform to the color separation conventions, a document must begin by using color separation convention operators and comments in its output, when appropriate. Details about individual operators and comments are provided below. Other guidelines and restrictions are outlined later in this document.

3.1 Convention Operators

The following "operators" are not defined in the *PostScript Language Reference Manual*, but should be used as pseudo-operators in your PostScript language output. Separation applications from Adobe Systems and other vendors will redefine these convention operators to separate your documents. Your application should conditionally define procedures with these special names, as shown later in this document.

findcmykcustomcolor *cyan magenta yellow black key* **findcmykcustomcolor** *array*

obtains and returns an array identifying a custom or process ink named *key* whose color approximately equals the process color specified by the tints *cyan*, *magenta*, *yellow*, and *black*, each of which must be a number between 0 and 1. *key* must be a string.

In normal use, the values *cyan, magenta, yellow,* and *black* in the array will be used as an approximation for printing on color printers. The *key* is used by a separation program to render items on the proper separation plane, at the tint specified with the **setcustomcolor** operator (defined below).

setcustomcolor *array tint* **setcustomcolor** —

sets the current color to the *tint* of the custom or process color specified by *array*. *tint* must be a number between 0 and 1, where a value of 0 corresponds to a 0% ink coverage, a value of 1 corresponds to a 100% ink coverage, and intermediate values correspond to intermediate coverages. *array* must be previously returned by the **findcmykcustomcolor** operator.

setseparationgray *gray* **setseparationgray** —

changes the current color to paint with a tint value of 1 - *gray* on all process and custom color plates. *gray* must be a number between 0 and 1. A tint (1 - *gray*) value of 0 corresponds to a 0% ink coverage, a tint value of 1 corresponds to a 100% ink coverage, and intermediate values correspond to intermediate coverages.

setseparationgray may be used to render graphics, such as registration and crop marks, that must appear on all separations.

customcolorimage *width height bits/sample matrix proc array* **customcolorimage** —

renders an image whose sample values specify the amount of the custom or process color identified by *array*, where an image sample value of 0 indicates 100% of the color, 1 indicates 0% of the color, and intermediate values correspond to intermediate ink coverage. *width*, *height*, *bits/sample*, *matrix* and *proc* are as defined for the multiple-argument version of the **image** operator. *array* must be previously returned by the **findcmykcustomcolor** operator.

separationimage *width height bits/sample matrix proc* **separationimage** —

renders an image on all process and custom color plates. *width, height, bits/sample, matrix* and *proc* are as defined for the multiple-argument version of the **image** operator. **separationimage** may be used to render graphics, such as registration and crop marks, that must appear on all separations.

setoverprint *boolean* **setoverprint** —

sets the value of the overprint parameter. If overprint is false, painting a shape causes knock-outs on all color planes (shape paints as white on all separations prior to rendering). If overprint is true, these knock-outs are not generated on unmarked color planes.

For example, when overprint is false, rendering a 50% magenta square will first result in all color planes being painted with a white square, then the magenta plane will be painted with a 50% tint square. When overprint is true, only the magenta color plane is painted with a white square prior to rendering the 50% tint square; no other color planes are affected.

Note *The overprint behavior described above, which applies to output produced by host-based separation applications, differs from the default overprint behavior on a Level 2 RIP. On the host, when marking with a multi-component color operator, such as* **setcmykcolor***, non-zero ink component values are treated as "no paint." On the RIP, however, the non-zero ink values are interpreted as "paint white." For example, a prepress application understands the call "*`1 1 0 0 setcmykcolor`*" to mean, "paint 100% tint on the cyan and magenta color planes; do not paint on the yellow and black color planes." However, a Level 2 RIP would interpret "*`1 1 0 0 setcmykcolor`*" to mean, "paint 100% tint on the cyan and magenta color planes, and paint white on the yellow and black planes." Due to this variance in interpretation, overprint results will differ between Level 1 host-based separations and Level 2 in-RIP separations when all of the following are true:*

1. *the overprint parameter is set to* true,
2. *marking is made using a multi-component color operator (such as* **setcmykcolor***), and*
3. *one or more of the ink component values is zero.*

Applications can achieve consistent results between host-based and in-RIP separations by marking overprinted process color objects using the Separation *color space, marking each non-zero component ink one component at a time. For example, your Level 1-style code path to fill a rectangle with process color may result in the following:*

```
gsave
true setoverprint
1 1 0 0 setcmykcolor
0 0 100 100 rectfill % or Level 1 equivalent
grestore
```

The Level 2 code path for the same operation may redefine some operators, so that the end result would be as follows:

```
gsave
true setoverprint
[/Separation (Cyan) /DeviceCMYK {0 0 0}] setcolorspace 1 setcolor
0 0 100 100 rectfill
[/Separation (Magenta) /DeviceCMYK {0 exch 0 0}] setcolorspace 1
  setcolor
0 0 100 100 rectfill
grestore
```

currentoverprint **currentoverprint** *boolean*

returns the current value of the overprint parameter.

setcmykoverprint *cyan magenta yellow black* **setcmykoverprint** —

sets the current color overprinting characteristics for the process colors *cyan*, *magenta*, *yellow*, and *black*, respectively. These values are numbers between 0 and 1, corresponding to the arguments to the **setcmykcolor** operator, or the special value -1.

If *cyan*, *magenta*, *yellow*, and/or *black* equals -1, **setcmykoverprint** overprints painted areas on the (Process Cyan), (Process Magenta), (Process Yellow), and/or (Process Black) separations, respectively, when the overprint parameter is *true* (set by **setoverprint**). If overprint is *false*, **setcmykoverprint** knocks out these areas with a tint of 0.

Note *The* **setcmykoverprint** *operator has not been adopted by the industry. It remains on the list of color convention operators because it has some perceived usefulness, but it is not supported by any shipping host-based separation applications today.*

3.2 Other Operators Affecting Color Separation

In addition to the convention operators, a number of standard PostScript operators are used in the color separation process. These operators may be redefined by separation utilities. For example, a separation program may redefine painting operators such as **fill** and **stroke** in order to achieve correct results on Level 1 devices when overprinting inks. Below we list some of these color operators in order to clarify their expected behavior in a separation environment. When applicable, information is provided about any restrictions in how these operators may be used. The behavior of these operators outside a separation environment is described in Section 8 of the *PostScript Language Reference Manual, Second Edition.*

setgray *gray* **setgray** —

sets the current color to the process color defined by the (Cyan), (Magenta), (Yellow), and (Black) tints 0, 0, 0, and 1 - *gray*, respectively. *gray* must be a number between 0 and 1. Note that the inverse of *gray* is used since the **setcmykcolor** operator interprets 1 as black, but the **setgray** operator interprets 0 as black.

setrgbcolor *red green blue* **setrgbcolor** —

sets the current color to a process color that approximately equals the color described by the parameters *red*, *green*, and *blue*, each of which must be a number in the range 0 to 1.

Note: Use of the **setrgbcolor** *operator is discouraged when a specific device CMYK result is desired.*

sethsbcolor *hue saturation brightness* **sethsbcolor** —

sets the current color to a process color that approximately equals the color described by the parameters *hue*, *saturation*, and *brightness*, each of which must be a number in the range 0 to 1.

Note: Use of the **sethsbcolor** *operator is discouraged when a specific device CMYK result is desired.*

setcmykcolor *cyan magenta yellow black* **setcmykcolor** —

sets the current color to the process color defined by the (Cyan), (Magenta), (Yellow), and (Black) tints cyan, magenta, yellow, and black, respectively, each of which must be a number between 0 and 1.

image *w h bits/sample matrix proc* **image** —

dict **image** —

renders an image whose sample values specify the tint of the process ink (Black). In the rendered area, **image** sets the tints of the process inks (Cyan), (Magenta), and (Yellow) to 0 (white).

In both the multi-operand and single-operand forms of the operator, the operands are as described in the *PostScript Language Reference Manual, Second Edition*. Note, however, that only the DeviceGray case of the single-operand version is currently supported by standard prepress applications.

colorimage *w h bits/sample matrix proc1 proc2 proc3 proc4 true 4* **colorimage** —

renders an image whose sample values specify the tints of the process inks (Process Cyan), (Process Magenta), (Process Yellow), and (Process Black).

For reasons of efficiency, only the multi-procedure CMYK form of the **colorimage** operator is currently supported. The single- and multi-procedure RGB forms and the single-procedure CMYK forms are not supported. As a result, application software creating conforming PostScript language programs must perform the RGB to CMYK data conversion.

3.3 DSC Color Separation Convention Comments

In addition to using the operators listed above in your PostScript language output, your application should also conform to the DSC (Document Structuring Conventions). An application used to separate or import your PostScript language files may use the information provided by DSC comments. In its user interface, for example, it may present a list of colors in the document, to allow the user to select which color separations to print.

When using DSC comments such as `%%DocumentCustomColors:`, the strings (Cyan), (Magenta), (Yellow), and (Black) are reserved for the four process color inks cyan, magenta, yellow, and black, respectively. Custom inks are identified by any arbitrary string not equal to any of the four process ink names, such as (Adobe Red) or (Pantone 435).

Color Convention Promotion

Applications importing EPS files which use color should promote color name information appropriately. Any custom or process colors used in the imported EPS file should appear in the header comments of the document which contains the EPS file. For example, if the document uses only process black, but the EPS imported has the following header comment:

```
%%DocumentProcessColors: Cyan
%%DocumentCustomColors: (Custom Red)
%%CMYKCustomColor: 0 0.8 0.9 0 (Custom Red)
```

then, the header comments for the document containing the EPS should reflect that information by having the header comments as follows:

```
%%DocumentProcessColors: Black Cyan
%%DocumentCustomColors: (Custom Red)
%%CMYKCustomColor: 0 0.8 0.9 0 (Custom Red)
```

If the definitions for custom colors are not consistent between the container document and the EPS file, the application should attempt to resolve that conflict. For example, Adobe PageMaker prompts the user and does not allow two colors used in one document to have conflicting definitions.

Color Header Comments

%%CMYKCustomColor: <CMYKcolor> …

<CMYKcolor> ::= <cya> <mag> <yel> <blk> <colorname>

<cya> :: = *<real>*	(Cyan percentage)
<mag> ::= *<real>*	(Magenta percentage)
<yel> ::= *<real>*	(Yellow percentage)
<blk> ::= *<real>*	(Black percentage)
<colorname> ::= *<text>*	(Custom color name)

This comment provides an *approximation* of the custom color specified by *colorname*. The four components of cyan, magenta, yellow, and black must be specified as numbers from 0 to 1 representing the percentage of that process color. The numbers are similar to the arguments to the **setcmykcolor** operator. The *colorname* follows the same custom color naming conventions as the %%DocumentCustomColors: comment.

%%DocumentCustomColors: { <colorname> ... } | (atend)

<colorname> ::= *<text>* (Custom color name)

This comment indicates the use of custom colors in a document. An application arbitrarily names these colors, and their CMYK or RGB approximations are provided through the %%CMYKCustomColor: or %%RGBCustomColor: comments in the body of the document. Normally, the *colorname* specified can be any arbitrary string except Cyan, Magenta, Yellow, or Black. If imaging to a specific process layer is desired, these names may be used.

%%DocumentProcessColors: { <color> ... } | (atend)

<color> ::= Cyan | Magenta | Yellow | Black

This comment marks the use of process colors in the document. Process colors are defined to be Cyan, Magenta, Yellow, and Black. This comment is used primarily when producing color separations. See also %%PageProcessColors:.

%%RGBCustomColor: <RGBcolor> …

<RGBcolor> ::= <red> <green> <blue> <colorname>

<red> ::= *<real>*	(Red percentage)
<green> ::= *<real>*	(Green percentage)
<blue> ::= *<real>*	(Blue percentage)
<colorname> ::= *<text>*	(Custom color name)

This comment provides an *approximation* of the custom color specified by *colorname*. The three components of red, green, and blue must be specified as numbers from 0 to 1 representing the percentage of that process color. The numbers are similar to the arguments to the **setrgbcolor** operator. The *colorname* follows the same custom color naming conventions as the %%DocumentCustomColors: comment.

Color Page Comments

%%PageCustomColors: **{ <colorname> ... } | (atend)**
<*colorname*> ::= <*text*> (Custom color name)

This comment indicates the use of custom colors in the page. An application arbitrarily names these colors, and their CMYK or RGB approximations are provided through the %%CMYKCustomColor: or %%RGBCustomColor: comments in the header section of the document. See the %%DocumentCustomColors: comment.

%%PageProcessColors: **{ <color> ... } | (atend)**
<*color*> ::= Cyan | Magenta | Yellow | Black

This comment marks the use of process colors in the page.
Process colors are defined as Cyan, Magenta, Yellow, and Black.
See the %%DocumentProcessColors: comment.

4 Color Separation Guidelines and Restrictions

Beyond using the color separation convention operators and DSC color comments, your PostScript language programs should adhere to some additional guidelines to work correctly with separation applications. Following the recommendations below should ensure successful separation of your composite output.

4.1 Specific Requirements

1. Do not retrieve operator definitions from systemdict.

   Separation programs will redefine the operators and convention operators listed in sections 3.1 and 3.2 of this document, as well as the painting operators such as **fill**, **stroke**, and **show** in order to create separations from your composite job. For that reason, it is imperative that you use the **load** operator when retrieving operator definitions from the dictionary stack, rather than explicitly retrieving definitions from systemdict.

   Do this:

   ```
   /f /fill load def
   ```

   not this:

   ```
   /f systemdict /fill get def
   ```

2. Conditionally define color separation convention operators, so your job will print properly in an in-RIP separation environment or when printed as a color composite file. Section 5.1 of this document offers sample code for doing this.

3. Conditionally define Level 2 and color extension operators.

   A few of the PostScript operators used for describing color documents are not supported by Level 1 PostScript output devices. For example, **setcmykcolor** and **colorimage** are not available on early Level 1 black and white devices. If your application or driver uses one of these operators, its code should define these operators to prevent receiving an undefined error at print time, which can happen if a separation environment doesn't provide a definition for these operators either. The definitions should be made conditionally to allow a separation environment to redefine the operators.

   Using the sep_ops resource code in example 5.1 of this document will prevent an undefined error with the **setcmykcolor** operator. Sample code is offered in section 5.3, which demonstrates how to ensure that **colorimage** is only called in environments that support that color extension.

4. Provide the user with the option to print using Level 1 painting operators only.

   Currently, host-based separation programs do not properly separate files containing Level 2 painting operators, such as **rectfill**. Separation programs should revise their code to add such support in the near future. In order to have your output work correctly with today's applications, you should offer your users an output path in which only Level 1 painting operators are used. Code using Level 2 painting operators may appear to separate correctly in host-based separation programs, but features such as overprinting may not work correctly. The exception to this rule is the **colorimage** operator and the dictionary form of the **image** operator for grayscale images; these should both work correctly, as described in section 3.2 of this document.

   Even though host-based separation is currently limited to Level 1 painting, recall that jobs containing any valid PostScript language code can be separated in the RIP on Level 2 imagesetters. For this reason, and because future prepress applications will add support for Level 2 operators, we recommend that you also build Level 2 painting operator support into your program now, as a user-selectable option. In that way, your users can choose this option when printing to Level 2 imagesetters, and with host-based separation programs in the future.

5. Provide user option to specify colors in DeviceCMYK or DeviceGray.

   As with guideline 4 above, limitations of today's host-based applications require that your code limit its color space usage to Level 1 color spaces;

specifically you should use the device-dependent color spaces: DeviceCMYK and DeviceGray for compatibility with host-based separation applications.

As with the Level 2 painting operators, we recommend that you provide your users the option to specify and output colors using Level 2 device-independent CIE color spaces. This will allow your users to achieve accurate color matching when printing composite files or making color separations in-RIP.

6. Do not use restricted graphic state operators.

   During the rendering of a host-based separation, the current color and the halftone screen graphic state parameters need to be controlled in precise ways. In order to simplify the implementation, the conventions require that a PostScript language program not use the **currentgray**, **currentrgbcolor**, **currenthsbcolor**, **currentcmykcolor**, **setscreen**, **currentscreen**, **setcolorscreen**, **currentcolorscreen**, or **initgraphics** operators. Note especially that this restriction precludes the use of **setscreen** and **setcolorscreen** as an implementation for rendering pattern fills.

7. Concatenate your transfer function to the current transfer function.

   Page descriptions should *not* contain transfer function modifications. However, if your code does use a transfer function in a page description, the function should be concatenated to the current transfer function as shown below. Doing this allows your page description to properly inherit any transfer effects that are introduced later in the prepress workflow, such as an inverse transfer to achieve negatives.

   Do this:

   ```
   [{… your transfer proc …} /exec cvx currenttransfer /exec cvx]
     cvx settransfer
   ```

   instead of:

   ```
   {… your transfer proc …} settransfer
   ```

8. Name custom colors and inks appropriately.

   Each custom ink in a PostScript language program should be identified by a distinct string. The strings (Cyan), (Magenta), (Yellow), and (Black), are reserved for the four process inks cyan, magenta, yellow, and black, respectively. Custom inks are identified by any arbitrary string not equal to any of the four process ink names, such as (Adobe Red) or (Pantone 435).

   Color names are case sensitive, therefore (Green) ≠ (green).

Note: Some applications currently on the market may also treat the following names as process colors: (Process Cyan), (Process Magenta), (Process Yellow), (Process Black). Application developers may wish to discourage users from using the above color names when custom ink, rather than process ink, is indicated.

9. Reset overprint parameter after **grestore** or **restore**.

   In PostScript language Level 2, overprint is introduced as a parameter in the graphics state. However, there is no overprint parameter or functionality in Level 1, and most host-based separation programs do not treat overprint as part of the graphic state. For that reason, after a **grestore** or **restore** call your code should explicitly set the overprint parameter to its value at the most recent **gsave** or **save**, respectively. By so doing, your overprint parameter will have the same value whether your program is executing in a Level 1 host-based separation environment or a Level 2 environment.

   Similarly, after calling **grestoreall**, your program should reset the overprint parameter to match the graphic state reset by the **grestoreall** call, to make the results consistent in Level 1 and Level 2 environments. The code below shows an example of how you may reset the overprint parameter.

Example 3: *Resetting overprint parameter*

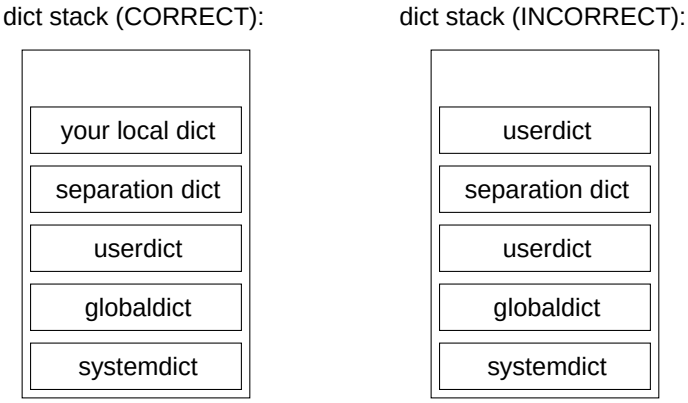
```
…
false setoverprint
gsave                  % position 1
   true setoverprint
   gsave               % position 2
      1 0 0 0 setcmykcolor
      false setoverprint
      drawbox
   grestore
   true setoverprint   % explicitly set overprint to value at
                       % last gsave (position 2)
   gsave
       0 1 1 0 setcmykcolor
grestoreall
false setoverprint  % explicitly set overprint to value at
                    % pos. 1 (the gstate restored by grestoreall)
…
```

10. Do not leave copies of userdict on the dictionary stack.

    Use **put** and **get** to retrieve values from userdict, or use a temporary dictionary to track variable values, as shown in code samples later in this document.

Following this guideline allows color separation programs to redefine operators in their own dictionary, which is necessary for host-based color separations to work. If your code leaves userdict on the dictionary stack and another utility modifies any of the color operator definitions in userdict, then the new userdict definition for the color operator will be used rather than the one intended by the separation utility. Figure 2 shows how the dictionary stack should be structured when your program is being separated. On the left is the correct behavior.

Figure 2 *Dictionary Stack in Separation Environment*



4.2 More Free Advice

1. This document only describes how to create composite EPS files intended for separation. Some applications, such as Adobe TrapWise, will accept multi-page composite files to be separated, and impose further restrictions on the file's content and structure. For information on creating multi-page files that work properly with such programs, contact the Adobe Developers Association.

2. Do not rely on the initial graphic state having a default current color value of "black". Your code should explicitly call "0 **setgray**" or "0 0 0 1 **setcmykcolor**" to set the current color to black. This will allow separation applications to properly separate black objects in your document, by redefining **setgray** and **setcmykcolor**.

3. Use of the **setrgbcolor** or the **sethsbcolor** operators is discouraged when specific CMYK results are desired; in such cases the user should specify desired colors in CMYK, or the application should convert RGB or HSB colors to CMYK and use the **setcmykcolor** operator. By so doing, your application will have control over the RGB-to-CMYK or HSB-to-CMYK mapping, and thus greater influence over the final results. In a Level 2 in-RIP environment, you should allow your users to specify colors in the device-independent CIE color spaces, to achieve accurate color matching in their output.

5 Examples

Below are some sample programs which demonstrate conformance to the color separation conventions. This code and sample output is in the PostScript software development kit (SDK), available from the Adobe Developers Association.

5.1 sep_ops ProcSet Resource

The sep_ops procset resource below contains sample code for conditionally defining each of the convention operators. You should use the procset below, or some equivalent, in your PostScript language output. Including convention operator definitions such as these will allow your composite EPS file to print correctly when sent directly to a printer, or in other environments where the convention operators are not defined. Composite files using the procset below should also separate correctly in-RIP, although overprint behavior will not be consistent with host-based prepress software in some cases. (See **setoverprint** description in section 3.1 of this document for more details.)

Example 4: *Sample code for conditionally defining color separation convention operators*

```
%%BeginResource: procset sep_ops 1.03 0
%%Title: (Separation Procs)
%%Version: 1.03 0
userdict /sep_ops 50 dict dup begin put

/bdef {bind def} bind def
/xdef {exch def} bdef
/colorimagebuffer {  % helper proc called by customcolorimage
   0 1 2 index length 1 sub {
      dup 2 index exch get 255 exch sub 2 index 3 1 roll put
   }for
}bdef

/addprocs { % {proc1} {proc2} addprocs {{proc1}exec {proc2} exec}
   [ 3 1 roll
   /exec load
   dup 3 1 roll
   ] cvx
} bdef

/L1? {
    /languagelevel where {
       pop languagelevel 2 lt
    }{
       true
    } ifelse
} bdef

/colorexists { % tests to see if printing on color device
    statusdict /processcolors known {
        statusdict /processcolors get exec
```

```
    }{ % processcolors not present
        /deviceinfo where { % check for dps environment
           pop deviceinfo /Colors known {
            deviceinfo /Colors get % get color value from DPS
            statusdict /processcolors {% add processcolors entry
                    deviceinfo /Colors known {
                    deviceinfo /Colors get
                 }{
                    1
                 } ifelse
              } put
              }{
                1
              } ifelse
         }{ % not in dps environment, assume monochrome
           1
         } ifelse
    } ifelse
    1 gt % return true for color devices, false for B&W
} bdef

/MakeReadOnlyArray    { % size => [array]
    /packedarray where {
       pop packedarray
    }{
       array astore readonly
    } ifelse
} bdef

/findcmykcustomcolor where {
    pop
}{
    /findcmykcustomcolor {% c m y k name findcmykcustomcolor array
       5 MakeReadOnlyArray
    } bdef
} ifelse

/setoverprint where {
    pop
}{
    /setoverprint {% boolean setoverprint -
       pop
    } bdef
} ifelse

/setcustomcolor where {
    pop
}{
    L1? {
       /setcustomcolor { % array tint setcustomcolor -
          exch
          aload pop pop
          4 { 4 index mul 4 1 roll } repeat
          5 -1 roll pop
          setcmykcolor
       } bdef
    }{
       /setcustomcolor { % customcolorarray tint
```

```
          exch
          [ exch /Separation exch dup 4 get exch /DeviceCMYK exch
            0 4 getinterval
            [ exch /dup load exch cvx {mul exch dup}
            /forall load /pop load dup] cvx
          ] setcolorspace setcolor
        } bdef
    } ifelse
} ifelse

% initialize variables to avoid unintentional early binding
/ik 0 def /iy 0 def /im 0 def /ic 0 def

/imagetint {% converts cmyk to grayscale equiv w/red book formula
             % called by setcmykcolor and customcolorimage procs.
    ic .3 mul
    im .59 mul
    iy .11 mul
    ik add add add dup
    1 gt{pop 1}if
} bdef

/setcmykcolor where {
    pop
}{
  % setcmykcolor not supported, call setgray instead
  /setcmykcolor { % c m y k setcmykcolor --
    /ik xdef /iy xdef /im xdef /ic xdef
    imagetint
    1 exch sub setgray
  } bdef
} ifelse


/customcolorimage where {
    pop
}{
   L1? {
       /customcolorimage{ % w h bps matrix proc array
          gsave
          colorexists {
            aload pop pop
            /ik xdef /iy xdef /im xdef /ic xdef
            currentcolortransfer
            {ik mul ik sub 1 add} addprocs
            4 1 roll {iy mul iy sub 1 add} addprocs
            4 1 roll{im mul im sub 1 add} addprocs
            4 1 roll{ic mul ic sub 1 add} addprocs
            4 1 roll setcolortransfer
            /magentabuf 0 string def
            /yellowbuf 0 string def
            /blackbuf 0 string def
            {
              colorimagebuffer dup length magentabuf length ne{
                 dup length dup dup
                 /magentabuf exch string def
                 /yellowbuf exch string def
                 /blackbuf exch string def
```

```
            }if
            dup magentabuf copy yellowbuf copy
            blackbuf copy pop
          } addprocs
          {magentabuf}{yellowbuf}{blackbuf} true 4 colorimage
        }{ % non-color device
          aload pop pop /ik xdef /iy xdef /im xdef /ic xdef
         /tint imagetint def
          currenttransfer
          {tint mul 1 tint sub add} addprocs settransfer image
        }ifelse
       grestore
     } bdef
  }{ % Level 2 environment
     /customcolorimage { % w h bps matrix proc array
         gsave
         [ exch /Separation exch dup 4 get exch /DeviceCMYK exch
           0 4 getinterval
           [ exch /dup load exch cvx {mul exch dup}
           /forall load /pop load dup] cvx
         ] setcolorspace

         10 dict begin
            /ImageType 1 def
            /DataSource exch def
            /ImageMatrix exch def
            /BitsPerComponent exch def
            /Height exch def
            /Width exch def
            /Decode [1 0] def
          currentdict end
          image
          grestore
     } bdef
   } ifelse
} ifelse

/setseparationgray where {
    pop
}{
    L1? {
        /setseparationgray {
           1 exch sub dup dup dup setcmykcolor
        } bdef
    }{
        /setseparationgray {
           [/Separation /All /DeviceCMYK
            {dup dup dup}] setcolorspace 1 exch sub setcolor
        } bdef
    } ifelse
} ifelse

/separationimage where {
    pop
}{
    /separationimage {
        gsave
        1 1 1 1 (All)
```

```
        findcmykcustomcolor customcolorimage
        grestore
      } bdef
} ifelse
currentdict readonly pop end
%%EndResource
```

5.2 Line Art Example

Type of Document	Line art containing CMYK and custom color objects
Color Separation Comments	%%DocumentProcessColors: %%DocumentCustomColors: %%CMYKCustomColor:
Convention Operators	**findcmykcustomcolor setcustomcolor, setoverprint**
Other Color Operators	**setcmykcolor**
Applications able to separate this code	FrameMaker 4 for Mac, Adobe PageMaker 6, Adobe TrapWise 2, Adobe PrePrint Pro 1, QuarkXPress 3, Adobe Separator 5, Macromedia FreeHand™ 5

This line art example uses spot and process colors to draw some simple shapes and text. The color separations of this file should be as follows: The word “Hello” should appear on the cyan plate. The word “World!” is screened back on the process black plate, along with a solid black rectangle, which is missing its bottom right corner. The Pantone Wm Red CV plate should have a 3-point stroked rectangle, with an ‘l’ character knocked out of it, and the Pantone Yellow CV plate should have a solid filled rectangle with the letters “llo” knocked out of it. Note that the string “World” does not knock out of any plate (ie. print as white on those plates) because it is set to overprint.

Example 5: *Code Sample demonstrating description of vector artwork*

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 141 407 279 629
%%Title: (my sample)
%%DocumentNeededResources: font Times-Roman
%%DocumentProcessColors: Cyan Black
%%DocumentCustomColors: (PANTONE Wm Red CV)
%%+ (PANTONE Yellow CV)
%%CMYKCustomColor: 0 0.79 0.91 0 (PANTONE Wm Red CV)
%%+ 0 0 1 0 (PANTONE Yellow CV)
%%Extensions: CMYK
%%EndComments
```

```
%%BeginProlog
%%BeginResource: procset sep_ops 1.0 0
… put sep_ops resource definition here …
%%EndResource
%%EndProlog

%%BeginSetup
sep_ops begin
50 dict begin % temp dict for variable definitions
%%EndSetup

%%Page: 1 1
%%BeginPageSetup
/pgsave save def
/Times-Roman findfont 24 scalefont setfont
0 0 0 1 setcmykcolor
%%EndPageSetup
220 485 moveto 220 628 lineto 142 628 lineto
142 485 lineto 220 485 lineto closepath

fill
0 0 1 0 (PANTONE Yellow CV) 0 % c m y k colorname tint
/tint exch def
findcmykcustomcolor
false setoverprint
tint 1 exch sub setcustomcolor

249 410 moveto 249 556 lineto 181 556 lineto
181 410 lineto 249 410 lineto closepath
gsave fill grestore

0 0.79 0.91 0 (PANTONE Wm Red CV) 0
/tint exch def
findcmykcustomcolor true setoverprint
tint 1 exch sub setcustomcolor
3 setlinewidth stroke
1 0 0 0 setcmykcolor false setoverprint
150 450 moveto (Hello ) show
0.5 setgray true setoverprint
(World!) show

pgsave restore
showpage
%%Trailer
end % temp dictionary
end % sep_ops
%%EOF
```

5.3 Image Examples

In addition to the examples below, a number of sample files which demonstrate the use of binary and encoded image data may be found in the latest version of the SDK.

CMYK Image Example

Type of Document	CMYK image
Color Separation Comments	%%DocumentProcessColors:
Convention Operators	(none)
Other Color Operators	**colorimage**
Applications able to separate this code	Adobe PageMaker 6, Adobe TrapWise 2, Adobe PrePrint Pro 1, QuarkXPress 3, Macromedia FreeHand 5

The following example paints an image of a blend (gradient) from process yellow (left) to process cyan (right). When separated, a blend appears on the process yellow plate, and a blend in the opposite direction appears on the cyan plate. Notice that when printing composites, the program does not call the **colorimage** operator in environments that do not support it, such as black and white Level 1 printers. The image data contains interleaved scan lines of cyan, magenta, yellow and black samples, followed by a scanline for a grayscale version of the image. In the event that **colorimage** is unknown, the grayscale image data is used with the multiple-argument form of the **image** operator.

Example 6: *CMYK image using either colorimage or image operator*

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100 100 300 300
%%Title: (Sample Color Image File)
%%DocumentProcessColors: Yellow Cyan
%%EndComments
%%BeginProlog
% define paintimage operator, conditional on environment

userdict /mydict 3 dict dup begin put
   /colorimage where { % colorimage is defined
       pop
       /paintimage {
           {currentfile cyanstr readhexstring pop}
           {currentfile magentastr readhexstring pop}
           {currentfile yellowstr readhexstring pop}
           {currentfile blackstr readhexstring pop
            currentfile graystr readhexstring pop pop}
            true 4 colorimage
       } bind def
```

```
   }{ % colorimage not defined (in L1 B&W environment)

       /paintimage {
          {currentfile cyanstr readhexstring pop pop
           currentfile magentastr readhexstring pop pop
           currentfile yellowstr readhexstring pop pop
           currentfile blackstr readhexstring pop pop
           currentfile graystr readhexstring pop}
          image
       } bind def
   } ifelse
currentdict readonly pop end
%%EndProlog
%%BeginSetup
mydict begin
50 dict begin % temp dict for EPS definitions
%%EndSetup

%%Page: 1 1
%%BeginPageSetup
/pgsave save def
0 setgray
%%EndPageSetup

/cyanstr 10 string def
/magentastr 10 string def
/yellowstr 10 string def
/blackstr 10 string def
/graystr 10 string def

100 100 translate
200 200 scale
10 10 8 [10 0 0 -10 0 10]
%%BeginData: 1068 Hex Bytes
paintimage
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
```

```
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
0019324B647D96AFC8E1 00000000000000000000
F9E0C7AE957C634A3118 00000000000000000000
E6E6DFD4CABEB2A69A90
%%EndData

pgsave restore
showpage
%%Trailer
end % temp dictionary
end % mydict
%%EOF
```

Monochrome Image Example

Type of Document	Monochrome image using spot color ink.
Color Separation Comments	%%DocumentCustomColors: %%CMYKCustomColor:
Convention Operators	**findcmykcustomcolor** **customcolorimage**
Other Color Operators	(none)
Applications able to separate this code	Adobe PageMaker 6, Adobe TrapWise 2, Adobe PrePrint Pro 1, QuarkXPress 3, Macromedia FreeHand 5

This example describes a document containing a gradient of the spot color Pantone Wm Red CV. When separations are produced, the gradient only appears on the custom plate. When custom colors are converted to their process color equivalents (an option provided by many prepress applications), the gradient appears on both the process yellow and process magenta plates, with appropriate tints applied.

Example 7: *Monochrome Image Example*

```
%%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100 100 300 300
%%Title: (custom color monotone image)
%%DocumentCustomColors: (PANTONE Wm Red CV)
%%CMYKCustomColor: 0 0.79 0.91 0 (PANTONE Wm Red CV)
%%Extensions: CMYK
%%EndComments

%%BeginProlog
%%BeginResource: procset sep_ops 1.0 0
… put sep_ops resource definition here …
%%EndResource
```

```
%%EndProlog

%%BeginSetup
sep_ops begin
50 dict begin % temp dict for EPS definitions
%%EndSetup

%%Page: 1 1
%%BeginPageSetup
/pgsave save def
0 setgray
%%EndPageSetup

/tempstr 10 string def
gsave
100 100 translate
200 200 scale
10 10 8 [10 0 0 -10 0 10]
{currentfile tempstr readhexstring pop}

0 0.79 0.91 0 (PANTONE Wm Red CV)
findcmykcustomcolor
%%BeginData: 226 Hex Bytes
customcolorimage
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
0019324B647D96AFC8E1
%%EndData
grestore

pgsave restore
showpage
%%Trailer
end % temp dict
end % sep_ops
%%EOF
```

5.4 Example of Printing on All Separations

Type of Document	document containing targets, crop marks, and gray ramp bar which print on all separations.
Color Separation Comments	%%DocumentProcessColors:
Convention Operators	**setseparationgray** **separationimage**
Other Color Operators	**setcmykcolor**
Applications able to separate this code	Adobe PageMaker 6, Adobe TrapWise 2, Adobe PrePrint Pro 1, QuarkXPress 3, Macromedia FreeHand 5

Certain objects or images, such as printer's marks, are printed on every separation in order to aid in aligning plates and calibrating the press. Below is an example of a document that puts printer's marks on each plate via the **setseparationgray** and **separationimage** convention operators. The document also contains some simple objects painted in the process inks cyan, magenta, yellow and black for testing purposes.

Example 8: *Example using setseparationgray and separationimage*

```
%%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 50 50 350 450
%%Title: (sample using separationgray and separationimage)
%%DocumentProcessColors: Cyan Magenta Yellow Black
%%EndComments

%%BeginProlog
%%BeginResource: procset sep_ops 1.0 0
… put sep_ops resource definition here …
%%EndResource

%%BeginResource: procset print_marks 1.0 0
%%Title: (Some Printer's Marks)
%%Version: 1.0 0
userdict /print_marks 10 dict dup begin put
/DrawTarget { % x y -> --
    gsave
    translate
    0 setseparationgray
    0.2 setlinewidth
    newpath
    0 0 3 0 360 arc fill
    0 0 5 0 360 arc stroke
    90 rotate
    4 {
      1 setseparationgray 0 0 moveto 0 3 lineto stroke
      0 setseparationgray 0 3 moveto 0 8 lineto stroke
```

```
      90 rotate
    } repeat
    grestore
} bind def
/DrawCrops { % llx lly h w -> --
    gsave
    0.2 setlinewidth
    %1 1 1 1 setcmykcolor
    0 setseparationgray
    4 2 roll translate
    0 -7 moveto 0 -15 lineto -7 0 moveto -15 0 lineto
    dup 0 translate
    0 -7 moveto 0 -15 lineto 7 0 moveto 15 0 lineto
    exch 0 exch translate
    0 7 moveto 0 15 lineto 7 0 moveto 15 0 lineto
    neg 0 translate
    0 7 moveto 0 15 lineto -7 0 moveto -15 0 lineto
    stroke grestore
} bind def

/DrawBar { % x y  -> --
    gsave
    translate
    gsave
    110 10 scale
    11 1 8 [11 0 0 -1 0 1]
    {<001A334D668099B2CCE5FF>}
    separationimage
    grestore
    0.2 setlinewidth 0 setseparationgray
    0 0 moveto 110 0 lineto 0 10 rlineto
    -110 0 rlineto closepath stroke
    grestore
}  bind def
currentdict readonly pop end
%%EndResource
%%EndProlog

%%BeginSetup
sep_ops begin
print_marks begin
50 dict begin % temp dict for variable definitions
0 setgray
%%EndSetup

%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup

100 100 300 200 DrawCrops
80 250 DrawTarget
320 250 DrawTarget
200 80 DrawTarget
200 420 DrawTarget
145 440 DrawBar
```

```
1 0 0 0 setcmykcolor
100 400 moveto 150 0 rlineto 0 -100 rlineto -150 0 rlineto fill

0 1 0 0 setcmykcolor
300 100 moveto -50 0 rlineto 0 75 rlineto 50 0 rlineto fill
0 0 1 0 setcmykcolor
newpath
200 300 75 0 360 arc gsave fill grestore 0 0 0 1 setcmykcolor
stroke

pgsave restore
showpage
%%Trailer
end % temp dictionary
end % print_marks
end % sep_ops
%%EOF
```

Appendix A: Separation Program Compatibility Checklist

Below is a summary of application support for various types of color content. The information in the chart assumes that the PostScript page description uses DSC color convention comments and the convention operators, as shown in examples earlier in this paper. Sample code for the listed types of documents is available in the PostScript SDK, available from the Adobe Developers Association.

key: √ = supported X = not supported

Contents of PostScript Language File	Adobe PageMaker 6.0, Adobe TrapWise 2.5, Adobe PrePrint Pro 1.5, Macromedia FreeHand 5.0, QuarkXPress 3.3	Adobe Separator 5.0 FrameMaker 4.0 (Mac only)
Line art painted with process color	√	√
Line art painted with spot color	√	√
Line art painted w/RGB (or HSB)	√	√
Images w/CMYK data (interleaved by scanline).	√	X
Documents which specify printers marks or other objects using **setseparationgray** or **separationimage**.	√	X
Images w/RGB (or HSB) data	X	X
Process or Custom Color Monotone Images, using **customcolorimage** operator.	√	X

Appendix B: Advice to Separation Program Developers

Authors of software programs that support color separations should familiarize themselves with the content of this technical note.

As stated a number of times throughout this document, there are limitations in the capabilities of today's prepress and page layout applications which prevent other developers and users from taking full advantage of Level 2 features. Below are some changes you may make in your separation code to allow users to achieve better quality and more efficient output. We recommend that separation programs do the following:

1. Add support for level 2 in-RIP separations to your program.

   Example 1 of this document shows a simple example of how to create process color separations in the RIP. Adding support for custom colors and other color convention features may be fairly straightforward. As a starting place, the definitions for the convention operators in the sep_ops procset resource (section 5.1 of this paper) offer working definitions for the convention operators that may be used in an in-RIP separation environment. For example, the procset defines **setcustomcolor** to set a Separation color as follows:

```
/setcustomcolor { % customcolorarray tint
  exch
  [ exch /Separation exch dup 4 get exch /DeviceCMYK exch
   0 4 getinterval
  [ exch /dup load exch cvx {mul exch dup}
  /forall load /pop load dup] cvx
  ] setcolorspace setcolor
} bind def
```

   Adding support for in-RIP separations to your application provides several benefits to your end-users:

   • *Increased printing performance.*
   Only one copy of the composite file needs to be sent for the entire separation job.

• *Wider range of EPS support.*
Your application can import and separate any EPS file into process color separations; the users' import choices are not limited to only those files which conform to the color separation conventions set forth in this paper. Furthermore, any EPS file that contains DSC color header comments can be imported and separated into process and custom color separations.

• *Both performance and quality benefits of Level 2 features.*
When using in-RIP separations, your application can take advantage of Level 2 operators, such as **rectfill**, which offer improved performance. Your program can also use other Level 2 features such as painting in device independent color spaces, which will give your users better color matching.

2. Add redefinitions for level 2 painting operators to your Level 1-style separations.

   We strongly recommend that you modify your separation code to support composite PostScript language files that paint objects with level 2 painting operators, such as **rectfill** and **rectstroke**. If your application does not yet work with the single-operand form of the **image** operator, support for grayscale and CMYK images can be added fairly easily. This will benefit your customers by making your Level 1-style separations more flexible. For a complete list of painting operators, see the operator summary in Section 8.1 of the *PostScript Language Reference Manual, Second Edition*.

Appendix C: References

Adobe Systems Incorporated, *PostScript Language Reference Manual, (2nd Edition)*, Addison-Wesley Publishing Company, Inc., 1985.

Agfa Corporation, *An Introduction to Digital Color Prepress*, Volumes I and II, Agfa Corporation, 1991.

Foley, J. et all, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990. ISBN 0-201-12110-7.

Hunt, R., *The Reproduction of Colour in Photography, Printing, and Television*, Fountain Press, 1987. ISBN 0-85242-356-X.

International Paper Company, *Pocket pal*, 15th Edition, International Paper Company, Memphis, TN, 1992.

Judd, D.B., and Wyszecki, G., *Color in Business, Science, and Industry (3rd Edition)*, John Wiley & Sons, Inc., New York, 1975.

Molla, Dr. R. K., *Electronic Color Separation*, R.K. Printing & Publishing Company, Montgomery, WV, 1988.

Wyszecki G., and Stiles, W.S., *Color Science: Concepts and Methods, Quantitative Data and Formula*, (2nd Edition), John Wiley & Sons, Inc., New York, 1982.

Index

T

U