ULTRIX Worksystem Software

digital

Color Extensions for the Display PostScript® System

Order Number: AA–PANAA–TE

POSTSCRIPT®
LANGUAGE

COLOR EXTENSIONS

Order No. AA–PANAA–TE

ADOBE SYSTEMS
INCORPORATED

POSTSCRIPT Language Color Extensions

October 25, 1989

Copyright © 1988, 1989 by Adobe Systems Incorporated.
All rights reserved.

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated.

This document replaces the previous version dated October 3, 1988.

Contents

1 ABOUT THIS MANUAL

This document contains:

- A description of the extensions to the POSTSCRIPT® language that support new color functionality.
- Detailed information on the associated color operators.

Section 2 provides a general introduction to color functionality in the POSTSCRIPT language.

Section 3 discusses the extended color capabilities of the POSTSCRIPT language.

Section 4 contains an alphabetical listing of descriptions for all color operators that have been added to the POSTSCRIPT language.

2 ABOUT THE POSTSCRIPT LANGUAGE COLOR EXTENSIONS

The POSTSCRIPT language has been extended to provide more complete color functionality. This includes cyan-magenta-yellow-black (CMYK) color specification, black generation and undercolor removal functions, screen and transfer functions for four separate color components, and extension of the **image** concept to a **colorimage** operator that accepts multiple color components.

Earlier versions of the POSTSCRIPT language support color using the **setrgbcolor** and **sethsbcolor** operators, which enable the POSTSCRIPT interpreter to paint filled regions, strokes, image masks, and characters in color. On black-and-white machines, these operators generate an equivalent gray shade, which is printed or displayed.

To support color more fully, the POSTSCRIPT language has been extended to provide the following functions:

- Most significantly, multiple color images: the **colorimage** operator renders a multiple color image; it functions analogously to the the **image** operator, but uses red-green-blue (RGB) or cyan-magenta-yellow-black (CMYK) color input and generates full-color output.

- Halftone screen definitions: the **setcolorscreen** operator specifies halftone screen definitions for red, green, blue, and gray, or cyan, magenta, yellow, and black concurrently; it is the logical expansion of **setscreen** as it takes the same three arguments to define each screen for each printing ink.
- Color correction: the **setcolortransfer** operator sets the transfer function parameters for red, green, blue, and gray; it is an expansion of **settransfer** to four color components. The **setblackgeneration** operator provides a black generation function that establishes a black component from a cyan, magenta, and yellow specification. The **setundercolorremoval** operator provides undercolor removal from the cyan, magenta, and yellow components to compensate for the addition of black by the black generation function.
- CMYK color specification: the **setcmykcolor** operator allows the user to set the current color in the graphics state to a cyan-magenta-yellow-black color directly, bypassing the color correction operators.

The POSTSCRIPT language supports one-color, three-color, and four-color output devices. The color devices can be binary (one-bit-per-pixel per color component) or gray-scale (multiple-bits-per-pixel per color component, representing a range of intensities of each color component). A binary device uses halftoning to produce intermediate shades of its color components. If a device has eight-bits-per-pixel per component, it is called a *full gray-scale* device, and uses no halftoning. Devices with more than one and fewer than eight bits per pixel use a combination of built-in intensities and halftoning to produce the full range of desired shades of their color components. Three-color devices may be either red-green-blue (RGB), typically for displays and film recorders, or cyan-magenta-yellow (CMY) for printers. Four-color devices are cyan-magenta-yellow-black (CMYK) for color printers and color separation making devices.

The color operators described in this document are available in two forms. New versions of the POSTSCRIPT interpreter have these operators built in. For older versions of the POSTSCRIPT interpreter, a package of POSTSCRIPT language programs that emulate these operators will be available.

3 NEW FEATURES

The color extensions to the POSTSCRIPT language include the CMYK color specification, black generation, and undercolor removal for tunable conversion from RGB to CMYK color specification, and operators that expand the scope of screens, transfer functions, and images to full color input and output. These new color features are described conceptually in the sections below.

3.1 CMYK COLOR SPECIFICATION

Colors are formed either by adding light to black or by subtracting light from white. Computer displays and film recorders typically add colors, while printing inks typically subtract colors. These two methods for forming colors give rise to the two major complementary color specifications, the additive RGB specification and the subtractive CMYK specification.

Accordingly, a color component in these specifications either indicates the amount of light it reflects or the amount of light it absorbs. Each one of three standard printing process colors, *cyan*, *magenta*, and *yellow*, absorb one of the standard light components, *red*, *green*, and *blue*, respectively. *Black*, a fourth standard printing process color, absorbs all components of light. In the red-green-blue (RGB) color specification, each of its red, green, and blue components is associated with a real number between 0.0 and 1.0, inclusive, where 0.0 represents dark (no light) and 1.0 represents full light. In the cyan-magenta-yellow-black (CMYK) color specification, each of the four components are associated with real numbers between 0.0 and 1.0 inclusive, where 0.0 represents full light (no ink), and 1.0 represents dark (full ink).

The following equations demonstrate the relationship between the RGB and CMYK color specifications. Since cyan is the absence of red light,

cyan = 1.0 – red.

Similarly,

magenta = 1.0 – green

and

yellow = 1.0 – blue,

so that a color that is 0.2 red, 0.7 green, and 0.4 blue can also be expressed as 1.0 – 0.2 = 0.8 cyan, 1.0 – 0.7 = 0.3 magenta, and 1.0 – 0.4 = 0.6 yellow. To improve the fidelity of blacks and grays, a fourth process color, black, is often available on color printers. Just as red in the RGB specification is the opposite of cyan in the CMYK specification, a black value is the opposite to a POSTSCRIPT language gray value; that is,

black = 1.0 – gray.

3.2 BLACK GENERATION AND UNDERCOLOR REMOVAL

Logically, cyan, magenta, and yellow are all that are needed to generate a printing color completely. Thus an equal percentage of cyan, magenta, and yellow should create the equivalent percentage of black. In reality, colored printing inks do not mix perfectly, and such combinations often form dark brown shades instead. Thus, it is often desirable to substitute real black ink for the mixed-black portion of a color to obtain a truer color rendition on a printer.

Black generation is the process of calculating the amount of black to be used when trying to print a particular color. Undercolor removal is the process of reducing the amount of cyan, magenta, and yellow components to compensate for the amount of black that was added by the black generation. Flexibility in performing these functions is important for achieving good results under different printing conditions.

The **setblackgeneration** operator provides the functionality to generate extra black, no black, or a black value equal to all or a fraction of the minimum values of cyan, magenta and yellow (see Section 4). Its argument is a procedure that takes one

numeric argument, the minimum value of *user cyan*, *magenta*, and *yellow* color components, and returns a single numeric result, the *user black* value (where a user color component value is that specified in the POSTSCRIPT language program before application of the corresponding transfer function). This procedure is automatically applied whenever **setrgbcolor**, the three-color case of **colorimage** (see Section 3), or **sethsbcolor** specify a color. This *user black* value is then mapped to a *device black* value by applying the gray transfer function to its difference from 1.0 and subtracting the result from 1.0 (see **setcolortransfer** in Section 4). The black generation function is not applied when **setgray**, **setcmykcolor** (see Section 4), or the one- or four-color cases of **colorimage** specify colors. This computed black value is used only when outputting on four-color output devices.

The **setundercolorremoval** operator (see Section 4) provides functionality to remove some amount of color from each of the cyan, magenta, and yellow components. This amount could be exactly the same amount as was generated to make the black component, zero (so no color is removed from the cyan, magenta and yellow components), some fraction of the black amount, or even a negative amount. Like **setblackgeneration**, this operator permits considerable flexibility in color correction.

The argument to **setundercolorremoval** is a procedure that takes one numeric argument, the minimum value of *user cyan*, *magenta*, and *yellow* color components, and returns a single numeric result that is subtracted from each of these original user color components. This procedure is applied whenever **setrgbcolor**, the three-color case of **colorimage** (see Section 4), or **sethsbcolor** specify a color. After subtracting the value generated in the above mapping from the color components and resetting negative values to 0.0 and values greater than 1.0 to 1.0, each component is subtracted from 1.0 to yield *red*, *green*, and *blue* components. Each of these components is mapped into a *device color component* using its respective transfer function (see **setcolortransfer** in Section 4). Undercolor removal is not applied when **setgray**, **setcmykcolor**, or the one- or four-color cases of **colorimage** specify a color. Undercolor removal is used only when outputting on four-color devices.

The following equations define the complete color transformation process from RGB to CMYK. The values red_u, grn_u, blu_u, gry_u, cyn_u, mag_u, yel_u, and blk_u are the (input) values of *user red*, *green*, *blue*, *gray*, *cyan*, *magenta*, *yellow*, and *black*, respectively. *UCR()* is the undercolor removal function. *BG()* is the black generation function. *RedT*, *GrnT*, *BluT*, and *GryT* are the red, green, blue, and gray transfer functions (see **setcolortransfer** in Section 4), respectively. The values red_d, grn_d, blu_d, gry_d, cyn_d, mag_d, yel_d, and blk_d are the (output) values of *device red*, *green*, *blue*, *gray*, *cyan*, *magenta*, *yellow*, and *black*, respectively.

$cyn_u = 1.0 - red_u$
$mag_u = 1.0 - grn_u$
$yel_u = 1.0 - blu_u$
$gry_u = 1.0$
$blk_u = 0.0$
$k = Min(cyn_u, mag_u, yel_u)$
$u = UCR(k)$
$red_d = RedT(1.0 - Min(1.0, Max(0.0, cyn_u - u)))$
$grn_d = GrnT(1.0 - Min(1.0, Max(0.0, mag_u - u)))$
$blu_d = BluT(1.0 - Min(1.0, Max(0.0, yel_u - u)))$
$gry_d = GryT(1.0 - BG(k))$
$cyn_d = 1.0 - red_d$
$mag_d = 1.0 - grn_d$
$yel_d = 1.0 - blu_d$
$blk_d = 1.0 - gry_d$

3.3 DIRECT CMYK COLOR SPECIFICATION

For the most demanding cases, color matching can require more complicated methods than those described above. The **setcmykcolor** operator and the four-color case of the **colorimage** operator bypass the black generation and undercolor removal operations, allowing the knowledgeable user to specify the cyan, magenta, yellow, and black color components for a particular device. This operator provides no correction other than the transfer functions **setcolortransfer** specifies (see Section 4); the results are device dependent.

The following equations define the complete color transformation process for the **setcmykcolor** operator and the four-color

case of the **colorimage** operator. The values are as defined in Black Generation and Undercolor Removal above. These equations are equivalent to those given above except for the omission of the black generation and undercolor removal steps.

$red_u = 1.0 - cyn_u$
$grn_u = 1.0 - mag_u$
$blu_u = 1.0 - yel_u$
$gry_u = 1.0 - blk_u$
$red_d = RedT(red_u)$
$grn_d = GrnT(grn_u)$
$blu_d = BluT(blu_u)$
$gry_d = GryT(gry_u)$
$cyn_d = 1.0 - red_d$
$mag_d = 1.0 - grn_d$
$yel_d = 1.0 - blu_d$
$blk_d = 1.0 - gry_d$

3.4 COLOR SCREENS, TRANSFER FUNCTIONS, AND IMAGES

The operators **setcolorscreen**, **currentcolorscreen**, **setcolortransfer**, and **currentcolortransfer** provide an expansion of the operators **setscreen**, **currentscreen**, **settransfer**, and **currenttransfer**, respectively, by setting up a screen and a transfer function for each color component. The **colorimage** operator provides an expansion of the **image** operator to allow samples of one, three, or four color components. (See Section 4 for more details.)

3.5 THE COLORIMAGE OPERATOR

colorimage is the logical expansion of **image** to handle sampled images whose samples are composed of more than one color component. The initial arguments to **colorimage** are the same as those for **image**. The final arguments differ according to the number of color components per sample and according to the encoding method.

The arguments to **colorimage** are as follows:

width height bits/component matrix $proc_0$ [...$proc_{ncolors-1}$] multiproc ncolors

(See Section 5 for precise definitions of these arguments.) *ncolors* describes the number of color components in each sample. Legal values for *ncolors* are 1 (gray-level samples only), 3 (RGB samples), or 4 (CMYK samples). *multiproc* distinguishes between encoding methods. The possibilities for *multiproc* are *false* (single procedure, color components are bunched together) or *true* (multiple procedures, one per color, components are separated into strings of like colors).

The legal variations of *ncolors* and *multiproc* allow the following possibilities (where proc subscripts have been changed to words to indicate the purpose of each proc):

w h b/c matrix $proc_{gray}$ false 1
w h b/c matrix $proc_{gray}$ true 1
w h b/c matrix $proc_{rgb}$ false 3
w h b/c matrix $proc_{red}$ $proc_{green}$ $proc_{blue}$ true 3
w h b/c matrix $proc_{cmyk}$ false 4
w h b/c matrix $proc_{cyan}$ $proc_{magenta}$ $proc_{yellow}$ $proc_{black}$ true 4

The first two variations here are both equivalent to

w h b/s matrix $proc_{gray}$ **image**

Data formats for colorimage operator. As indicated above, the **colorimage** operator has two forms, distinguished by its *multiproc* argument.

The single-procedure form is most useful if sample input is taken from a source that has already merged the color components. This form provides samples for which each RGB triple or CMYK quadruple is packed together in the string result of the procedure, using one of the following bit formats (where the high-order bit is shown on the left):

Bits/ comp.	RGB Format				
1	RGBRGBRG	BRGBRGBR	GBRGBRGB	RGBRGBRG	....
2	RRGGBBRR	GGBBRRGG	BBRRGGBB	RRGGBBRR	....
4	RRRRGGGG	BBBBRRRR	GGGGBBBB	RRRRGGGG	....
8	RRRRRRRR	GGGGGGGG	BBBBBBBB	RRRRRRRR	....

or

Bits/ comp.	CMYK Format				
1	CMYKCMYK	CMYKCMYK	CMYKCMYK	CMYKCMYK	....
2	CCMMYYKK	CCMMYYKK	CCMMYYKK	CCMMYYKK	....
4	CCCCMMMM	YYYYKKKK	CCCCMMMM	YYYYKKKK	....
8	CCCCCCCC	MMMMMMMM	YYYYYYYY	KKKKKKKK	....

The multiple-procedure form expects each procedure to return a string of values for only one color component per sample, using the same format as strings returned by the *proc* argument of the **image** operator. For a three-color image, $proc_0$ returns red values, $proc_1$ returns green values, and $proc_2$ returns blue values. For a four-color image, $proc_0$ returns cyan values, $proc_1$ returns magenta values, $proc_2$ returns yellow values, and $proc_3$ returns black values. The **colorimage** operator calls each of these procedures in turn, starting with $proc_0$ and continuing with $proc_1$, $proc_2$, and, if available, $proc_3$. When the *colorimage* operator needs more samples, it calls these procedures again in the same order. The color procedures must use separate strings for the three or four results of the three or four procedures, i.e., reusing the red string for the green values can result in some of the red values being lost. Also, the three or four procedures must return strings of identical lengths within each cycle of three or four calls.

The multiple-procedure form is most useful when color sample data are taken from separate color scanner passes. The **colorimage** operator requires the color data to be interleaved, as it must have all three or four components of any sample at the same time to do its work. The single-procedure form interleaves the data at the sample level; this may be convenient only if the data are already in that form when preparing the POSTSCRIPT language page description. The multiple-procedure form allows interleaving at a much coarser level. Typically, each procedure of the multiple-procedure form returns components for some number of scan lines of samples, where the number of components returned at each call is limited by the string storage available in the POSTSCRIPT interpreter.

Examples for colorimage operator. The following examples illustrate the use of the **colorimage** operator:

EXAMPLE 1:

```
/rgbstr 192 string def    % string to hold 256 two-bit samples
                          %   each of red, green, and blue data
45 140 translate          % locate lower left corner of image
132 132 scale             % map image to 132 point square
256 256 2                 % dimensions of source image
[256 0 0 –256 0 256]      % map unit square to source
{currentfile              % read image data from program file
  rgbstr readhexstring pop}
false 3                   % single proc, 3 colors, bit format:
                          %   rrggbbrr ggbbrrgg bbrrggbb ...
colorimage
94a1bec8c0b371a3a5c4d281 ...     (98304 hex digits of image data)
```

The above is a one-procedure 2-bit RGB image. The base-4 representation of the hex data is

2110 2201 2332 ...

which is composed of the following color samples:

r=2 g=1 b=1 r=0 g=2 b=2 r=0 g=1 b=2 r=3 g=3 b=2 ...

EXAMPLE 2:

```
/rstr 256 string def      % string to hold 256 8-bit red samples
/gstr 256 string def      % string to hold 256 8-bit green samples
                          %   (distinct from rstr)
/bstr 256 string def      % string to hold 256 8-bit blue samples
                          %   (distinct from rstr and bstr)
45 140 translate          % locate lower left corner of image
132 132 scale             % map image to 132 point square
256 256 8                 % dimensions of source image
[256 0 0 –256 0 256]      % map unit square to source
{currentfile rstr readhexstring pop}
                          % read red data from program file
{currentfile gstr readhexstring pop}
                          % read green data from program file
{currentfile bstr readhexstring pop}
                          % read blue data from program file
true 3                    % multiple proc, 3 colors
colorimage
7b5e606969615365556a6a66 ...     (512 hex digits of red data)
88868d848a92878578787a82 ...     (512 hex digits of green data)
62717c7b736e707d7b6a7c79 ...     (512 hex digits of blue data)
7d8b8d8c837d8b8e9284878e ...     (512 hex digits of red data)
2788b838b8e8e86868988908 ...     (512 hex digits of green data)
81817d857f85858290949487 ...     (512 hex digits of blue data)
... (390144 more hex digits of RGB data, cycling as above)
```

The above is a three-procedure 8-bit RGB image. The initial samples for each color, in hex, are

```
red:    7b  5e  60  69  ...
green:  88  86  8d  84  ...
blue:   62  71  7c  7b  ...
```

EXAMPLE 3:

```
/cstr 128 string def        % string to hold 256 4-bit cyan samples
/mstr 128 string def        % string to hold 256 4-bit magenta samples
                            %   (distinct from cstr)
/ystr 128 string def        % string to hold 256 4-bit yellow samples
                            %   (distinct from cstr and mstr)
/kstr 128 string def        % string to hold 256 4-bit black samples
                            %   (distinct from cstr, mstr, and ystr)
45 140 translate            % locate lower left corner of image
132 132 scale               % map image to 132 point square
256 256 4                   % dimensions of source image
[256 0 0 –256 0 256]        % map unit square to source
{currentfile cstr readhexstring pop}
                            % read cyan data from program file
{currentfile mstr readhexstring pop}
                            % read magenta data from program file
{currentfile ystr readhexstring pop}
                            % read yellow data from program file
{currentfile kstr readhexstring pop}
                            % read black data from program file
true 4                      % multiple proc, 4 colors
colorimage
e1d8caa57b655b6779606b72 ...    (256 hex digits of cyan data)
6bdbb867b9fb6a4859569989 ...    (256 hex digits of magenta data)
996796e639cc0b29f94736c7 ...    (256 hex digits of yellow data)
c9c0cad0d3cad2b7c9e2d7d8 ...    (256 hex digits of black data)
5d2d6d7d4d3d1d4d6c9d4d9d ...    (256 hex digits of cyan data)
4cdcfd4d6d1d8d7d5d4d2d2d ...    (256 hex digits of magenta data)
d2b7c9e2d7d8d8cbbac2d9d8 ...    (256 hex digits of yellow data)
88ae96632a70f6f4d8d9d9d8 ...    (256 hex digits of black data)
... (260096 more hex digits of CMYK data, cycling as above)
```

The above is a four-procedure four-bit CMYK image. The initial samples for each color, in hex, are

```
cyan:     e 1 d 8 c a a 5 ...
magenta:  6 b d b b 8 6 7 ...
yellow:   9 9 6 7 9 6 e 6 ...
black:    c 9 c 0 c a d 0 ...
```

EXAMPLE 4:

```
/cstr 1024 string def      % string to hold 1024 8-bit cyan samples
/mstr 1024 string def      % string to hold 1024 8-bit magenta samples
                           %   (distinct from cstr)
/ystr 1024 string def      % string to hold 1024 8-bit yellow samples
                           %   (distinct from cstr and mstr)
/kstr 1024 string def      % string to hold 1024 8-bit black samples
                           %   (distinct from cstr, mstr, and ystr)
/cfile (img/smp.c) (r) file def    % binary file containing 1048576 8-bit
                                   %   cyan samples
/mfile (img/smp.m) (r) file def    % binary file containing 1048576 8-bit
                                   %   magenta samples
/yfile (img/smp.y) (r) file def    % binary file containing 1048576 8-bit
                                   %   yellow samples
/kfile (img/smp.k) (r) file def    % binary file containing 1048576 8-bit
                                   %   black samples
36 126 translate           % locate lower left corner of image
540 540 scale              % map image to 540 point square
1024 1024 8                % dimensions of source image
[1024 0 0 -1024 0 1024]        % map unit square to source
{cfile cstr readstring pop}    % read cyan data from img/smp.c
{mfile mstr readstring pop}    % read magenta data from img/smp.m
{yfile ystr readstring pop}    % read yellow data from img/smp.y
{kfile kstr readstring pop}    % read black data from img/smp.k
true 4                     % multiple proc, 4 colors
colorimage
```

The above is a four-procedure 8-bit CMYK image, with cyan, magenta, yellow, and black samples taken from the files *img/smp.c*, *img/smp.m*, *img/smp.y*, and *img/smp.k*, respectively. This example only applies to a POSTSCRIPT interpreter that has a file system.

3.6 COLOR IMPLEMENTATIONS

Each POSTSCRIPT interpreter contains default color output methods that correspond to the printer to which it is attached. If the attached printer is a direct-color binary device, the standard output method produces three- or four-color output. If the attached printer is a gray-scale color device, then the POSTSCRIPT interpreter uses a gray-scale three- or four-color output method. If the attached printer is a black-and-white device, then the default output method produces a single black-and-white rendition of each page described.

Black-and-white printers also have the capability of producing color separations. These are three or four black-and-white pages for each page described. Each output page corresponds to the output for one color component. These separations are only useful if they are used in another printing process in which they are combined with color to form the intended full-color output.

The method by which a POSTSCRIPT language program informs the POSTSCRIPT interpreter that color separations are desired is still to be determined. This color separation capability will not be provided in the backward compatibility package, as it requires considerable internal changes in the POSTSCRIPT interpreter.

4 OPERATORS

The following pages contain an alphabetical listing of the new POSTSCRIPT color operators.

colorimage width height bits/component matrix $proc_0$ [...$proc_{ncolors-1}$] multiproc
ncolors **colorimage** –

renders a sampled image, whose samples contain one, three, or four color components on the current page. The first four arguments are the same as those for the **image** operator. The *bits/component* argument applies equally to all color components. **colorimage** permits its $proc_i$ arguments to return RGB or CMYK sample values rather than single-color (gray) values as are returned by the *proc* argument of the **image** operator.

The *ncolors* argument, a 1, 3, or 4, is the number of color components represented in the samples. If *ncolors* is 1, the samples have only one component, a gray component, and the operation of **colorimage** is equivalent to that of **image** with the same five initial arguments. If the *ncolors* argument is 3, the **colorimage** operator takes RGB (light-high) samples. If the *ncolors* argument is 4, the **colorimage** operator takes CMYK (dark-high) samples. On a four-color (CMYK) machine, the POSTSCRIPT interpreter converts a three-color (RGB) image to CMYK using the black generation and undercolor removal functions, whereas a four-color (CMYK) image bypasses these operations.

The *multiproc* argument is a boolean that distinguishes between two forms of the **colorimage** operator: *false* indicates the single-procedure form, argument ($proc_0$), and *true* indicates the multiple-procedure form, which requires one procedure argument per sample color ($proc_0$ ... $proc_{ncolors-1}$), i.e., three procedure arguments for RGB samples or four-procedure arguments for CMYK samples. If the *ncolors* argument is 1, then there is only one procedure argument, $proc_0$, regardless of the value of the *multiproc* argument. For a detailed description of the data formats and how the $proc_i$ procedures are called, see Data formats for colorimage Operator in Section 4, New Features.

Use of **setcolorscreen**, **setcolortransfer**, **setscreen**, or **settransfer** by any of the $proc_i$ procedures causes unpredictable results. Use of the **colorimage** operator after a **setcachedevice** within the context of a **BuildChar** procedure is not permitted (an **undefined** error results).

ERRORS:
limitcheck, rangecheck, stackunderflow, typecheck, undefined, undefinedresult

currentblackgeneration – **currentblackgeneration** proc

returns the current black generation function in the graphics state (see **setblackgeneration**).

ERRORS:
stackoverflow

currentcmykcolor – **currentcmykcolor** cyan magenta yellow black

returns the four components of the current color in the graphics state according to the cyan-magenta-yellow-black color specification (see **setcmykcolor**).

Note that the **currentgray** operator returns a weighted average of all four color components. Applying it is the equivalent of the following use of **currentcmykcolor**:

```
1.0 currentcmykcolor 4 1 roll 0.11 mul 3 1 roll 0.59 mul
exch 0.30 mul add add add sub dup 0.0 lt {pop 0.0} if
```

ERRORS:
stackoverflow

currentcolorscreen – **currentcolorscreen** r/c-frequency r/c-angle r/c-proc g/m-frequency g/m-angle g/m-proc b/y-frequency b/y-angle b/y-proc g/b-frequency g/b-angle g/b-proc

returns all 12 current halftone screen parameters in the graphics state (see **setcolorscreen**).

The **currentcolorscreen** operator is the logical expansion of **currentscreen** to four color components. Applying the **currentscreen** operator returns the three parameters describing the gray/black screen. It is the equivalent of the following use of **currentcolorscreen**:

```
currentcolorscreen 12 3 roll 9 {pop} repeat
```

ERRORS:
stackoverflow

currentcolortransfer – **currentcolortransfer** redproc greenproc blueproc grayproc

returns the current transfer functions in the graphics state for each of the four color components (see **setcolortransfer**).

The **currentcolortransfer** operator is the logical expansion of **currenttransfer** to four color components. Applying the **currenttransfer** operator returns the gray transfer function. It is the equivalent of the following use of **currentcolortransfer**:

```
currentcolortransfer 4 1 roll pop pop pop
```

ERRORS:
stackoverflow

currentundercolorremoval – **currentundercolorremoval** proc

returns the current undercolor removal function in the graphics state (see **setundercolorremoval**).

ERRORS:
stackoverflow

setblackgeneration proc **setblackgeneration** –

sets the current black generation function parameter in the graphics state. The *proc* operand must be a POSTSCRIPT language procedure that can be called with a number in the range 0.0 to 1.0 (inclusive) on the operand stack and which returns a number in the same range. This procedure maps the minimum of the *user cyan*, *magenta*, and *yellow* color components to *user black* values.

For additional information, see section 3.

EXAMPLE:

```
{dup .75 le {pop 0.0} {.75 sub 4.0 mul} ifelse} setblackgeneration
```

This sets the black component to zero when the minimum of cyan, magenta, and yellow is less than or equal to .75, while minima greater than .75 produce a black component that increases linearly from 0.0 (at a minimum of .75) to 1.0 (when user cyan, magenta, and yellow all have values of 1.0).

The use of **setblackgeneration** after a **setcachedevice** operation within the scope of a **BuildChar** procedure is not permitted (an **undefined** error results).

ERRORS:
stackunderflow, typecheck

setcmykcolor cyan magenta yellow black **setcmykcolor** –

sets the current color parameter in the graphics state to a color described by the parameters *cyan*, *magenta*, *yellow*, and *black*, each of which must be a number in the range 0.0 to 1.0. This establishes the color used subsequently to paint shapes such as lines, areas, and characters on the current page. This operator bypasses the black generation and undercolor removal operations.

For additional information, see section 3.

Note that applying the **setgray** operator sets the gray color component to its single argument value and the red, green, and blue color components to 1.0. It is the equivalent of the following use of **setcmykcolor**:

```
0.0 0.0 0.0 1.0 5 –1 roll sub setcmykcolor
```

The use of **setcmykcolor** after a **setcachedevice** operation within the scope of a **BuildChar** procedure is not permitted (an **undefined** error results).

ERRORS:
stackunderflow, typecheck

setcolorscreen r/c-frequency r/c-angle r/c-proc g/m-frequency g/m-angle g/m-proc b/y-frequency b/y-angle b/y-proc g/b-frequency g/b-angle g/b-proc **setcolorscreen** –

sets the current halftone screen definitions for red/cyan (r/c), green/magenta (g/m), blue/yellow(b/y), and gray/black (g/b) output color components in the graphics state. Each of the *r/c-frequency*, *g/m-frequency*, *b/y-frequency*, and *g/b-frequency* operands is a number that specifies the screen frequency for one output color component, measured in halftone cells per inch in device space. The *r/c-angle*, *g/m-angle*, *b/y-angle*, and *g/b-angle* operands specify the number of degrees by which their respective halftone screens are rotated with respect to the device coordinate system. The *r/c-proc*, *g/m-proc*, *b/y-proc*, and *g/b-proc* operands are each a POSTSCRIPT language procedure (as for **setscreen**). They define one color component's spot function to determine the order in which pixels within a halftone cell of that color component are assigned a light value to produce any desired shade of that color component. As with **setscreen**, the pixels with the highest values returned by this function become dark soonest at the lightest shades of that color, and the pixels with the lowest values becomes dark latest at the darkest shades of that color. The red/cyan, green/magenta, and blue/yellow screens have no effect on a black-and-white device, the gray/black screen has no effect on an RGB or CMY device, and no screens have any effect on a full (8-bits-per-pixel) gray-scale device.

Many binary color printers (those that print each dot at full intensity or not at all) require different angles for each color-component halftone to have attractive output. Each color printer containing a POSTSCRIPT interpreter has a default color screen chosen to look good on that printer.

The **setcolorscreen** operator is the logical expansion of **setscreen** to four color components. It takes the same three types of arguments as **setscreen**, but repeated four times. Applying the **setscreen** operator sets all four screens equally. It is the equivalent of the following use of **setcolorscreen**:

```
3 copy 6 copy setcolorscreen
```

EXAMPLE:

```
% 50 line dot screen with 75 degree cyan,
% 15 degree magenta,
% 0 degree yellow, and 45 degree black angled screens,
% which are standard for color printing
/sfreq 50 def              % 50 halftone cells per inch
/sproc {dup mul exch dup mul add 1 exch sub} def
                           % dot-screen spot function
sfreq 75 /sproc load       % 75 degree red (cyan) screen
sfreq 15 /sproc load       % 15 degree green (magenta) screen
sfreq 0 /sproc load        % 0 degree blue (yellow) screen
sfreq 45 /sproc load       % 45 degree gray (black) screen
setcolorscreen
```

ERRORS:
limitcheck, rangecheck, stackunderflow, typecheck

setcolortransfer redproc greenproc blueproc grayproc **setcolortransfer** –

sets the current transfer function parameters for red, green, blue, and gray in the graphics state. Each operand must be a POSTSCRIPT language procedure that may be called with a number in the range 0.0 to 1.0 (inclusive) on the operand stack and which will return a number in the same range. These procedures map user values of the color components (e.g., those specified by **setrgbcolor** and adjusted by **setblackgeneration** and **setundercolorremoval**, or 1.0 minus those specified by **setcmykcolor**) to *device* color components (for halftones, a weighted average of the lightness of pixels in a halftone cell). Only those transfer functions corresponding to color components supported by a device will have an effect on that device's output. For example, *redproc*, *greenproc*, and *blueproc* will have no effect on a black-and-white device, while *grayproc* will have no effect on an RGB device.

The single-color **settransfer** operator takes a single procedure argument whose purpose is to provide gamma correction for a printer's halftoning response. That operator is useful for a variety of effects beyond its original intention as a gray response correction function, but it is useful only in the context of a single output color, as on black-and-white printers. The **setcolortransfer** operator is the logical expansion of **settransfer** to four color components; it takes four function arguments, each similar in purpose to the function argument of **settransfer**, but each function separately controls the response for each of the red (1.0 minus cyan), green (1.0 minus magenta), blue (1.0 minus yellow) and gray (1.0 minus black) components, respectively.

Applying the **settransfer** operator sets all four transfer functions equally. It is the equivalent of the following use of **setcolortransfer**:

```
dup dup dup setcolortransfer
```

EXAMPLE:

```
{} {} {dup mul} {} setcolortransfer
```

This sets device blue as the square of user blue and leaves the other color components unchanged.

Calling **settransfer** with the argument

```
{1 exch sub}
```

to invert an output image is not guaranteed to work if any of the following operators are used in generating the image: **colorimage**, **setcmykcolor**, **setcolortransfer**, **sethsbcolor**, and **setrgbcolor**. In the case of a device with four color components, inversion can be more complicated than merely inverting all of the components.

The use of **setcolortransfer** after a **setcachedevice** operation within the scope of a **BuildChar** procedure is not permitted (an **undefined** error results).

ERRORS:
stackunderflow, typecheck

setundercolorremoval proc **setundercolorremoval** –

sets the current undercolor removal function parameter in the graphics state. The *proc* operand must be a POSTSCRIPT language procedure that may be called with a number in the range 0.0 to 1.0 (inclusive) on the operand stack and which will return a number in the range –1.0 (to *increase* the color components) to +1.0 (to *decrease* the color components). This procedure maps the minimum of the *user cyan*, *magenta*, and *yellow* color components to a value to be subtracted from each of these same components.

For additional information, see section 3.

EXAMPLE:

```
{currentblackgeneration exec .5 mul} setundercolorremoval
```

This sets the undercolor removal to be half of the black component from black generation.

The use of **setundercolorremoval** after a **setcachedevice** operation within the scope of a **BuildChar** procedure is not permitted (an **undefined** error results).

ERRORS:
stackunderflow, typecheck

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	———————	Local DIGITAL subsidiary or approved distributor
Internal[1]	———————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).