CHAPTER 4

Graphics

The graphics operators used in PDF content streams describe the appearance of pages that are to be reproduced on a raster output device. The facilities described in this chapter are intended for both printer and display applications.

The graphics operators form six main groups:

- *Graphics state operators* manipulate the data structure called the *graphics state*, the global framework within which the other graphics operators execute. The graphics state includes the *current transformation matrix* (CTM), which maps user space coordinates used within a PDF content stream into output device coordinates. It also includes the *current color*, the *current clipping path*, and many other parameters that are implicit operands of the painting operators.
- *Path construction operators* specify *paths*, which define shapes, line trajectories, and regions of various sorts. They include operators for beginning a new path, adding line segments and curves to it, and closing it.
- *Path-painting operators* fill a path with a color, paint a stroke along it, or use it as a clipping boundary.
- *Other painting operators* paint certain self-describing graphics objects. These include sampled images, geometrically defined shadings, and entire content streams that in turn contain sequences of graphics operators.
- *Text operators* select and show *character glyphs* from *fonts* (descriptions of typefaces for representing text characters). Because PDF treats glyphs as general graphical shapes, many of the text operators could be grouped with the graphics state or painting operators. However, the data structures and mechanisms for dealing with glyph and font descriptions are sufficiently specialized that Chapter 5 focuses on them.

- *Marked-content operators* associate higher-level logical information with objects in the content stream. This information does not affect the rendered appearance of the content (although it may determine if the content should be presented at all; see Section 4.10, "Optional Content"); it is useful to applications that use PDF for document interchange. Marked content is described in Section 10.5, "Marked Content."

This chapter presents general information about device-independent graphics in PDF: how a PDF content stream describes the abstract appearance of a page. *Rendering*—the device-dependent part of graphics—is covered in Chapter 6. The Bibliography lists a number of books that give details of these computer graphics concepts and their implementation.

4.1 Graphics Objects

As discussed in Section 3.7.1, "Content Streams," the data in a content stream is interpreted as a sequence of *operators* and their *operands*, expressed as basic data objects according to standard PDF syntax. A content stream can describe the appearance of a page, or it can be treated as a graphical element in certain other contexts.

The operands and operators are written sequentially using postfix notation. Although this notation resembles the sequential execution model of the PostScript language, a PDF content stream is not a program to be interpreted; rather, it is a static description of a sequence of *graphics objects*. There are specific rules, described below, for writing the operands and operators that describe a graphics object.

PDF provides five types of graphics objects:

- A *path object* is an arbitrary shape made up of straight lines, rectangles, and cubic Bézier curves. A path may intersect itself and may have disconnected sections and holes. A path object ends with one or more painting operators that specify whether the path is stroked, filled, used as a clipping boundary, or some combination of these operations.
- A *text object* consists of one or more character strings that identify sequences of glyphs to be painted. Like a path, text can be stroked, filled, or used as a clipping boundary.

- An *external object (XObject)* is an object defined outside the content stream and referenced as a named resource (see Section 3.7.2, "Resource Dictionaries"). The interpretation of an XObject depends on its type. An *image XObject* defines a rectangular array of color samples to be painted; a *form XObject* is an entire content stream to be treated as a single graphics object. Specialized types of form XObjects are used to import content from one PDF file into another *(reference XObjects)* and to group graphical elements together as a unit for various purposes *(group XObjects)*. In particular, the latter are used to define *transparency groups* for use in the transparent imaging model (*transparency group XObjects*, discussed in detail in Chapter 7). There is also a *PostScript XObject*, whose use is discouraged.
- An *inline image object* uses a special syntax to express the data for a small image directly within the content stream.
- A *shading object* describes a geometric shape whose color is an arbitrary function of position within the shape. (A shading can also be treated as a color when painting other graphics objects; it is not considered to be a separate graphics object in that case.)

PDF 1.3 and earlier versions use an *opaque imaging model* in which each graphics object is painted in sequence, completely obscuring any previous marks it may overlay on the page. PDF 1.4 introduces a *transparent imaging model* in which objects can be less than fully opaque, allowing previously painted marks to show through. Each object is painted on the page with a specified *opacity*, which may be constant at every point within the object's shape or may vary from point to point. The previously existing contents of the page form a *backdrop* with which the new object is *composited*, producing results that combine the colors of the object and backdrop according to their respective opacity characteristics. The objects at any given point on the page can be thought of as forming a *transparency stack*, where the stacking order is defined to be the order in which the objects are specified, bottommost object first. All objects in the stack can potentially contribute to the result, depending on their colors, shapes, and opacities.

PDF's graphics parameters are so arranged that objects are painted by default with full opacity, reducing the behavior of the transparent imaging model to that of the opaque model. Accordingly, the material in this chapter applies to both the opaque and transparent models except where explicitly stated otherwise; the transparent model is described in its full generality in Chapter 7.

Although the painting behavior described above is often attributed to individual operators making up an object, it is always the object as a whole that is painted. Figure 4.1 shows the ordering rules for the operations that define graphics objects. Some operations are permitted only in certain types of graphics objects or in the intervals between graphics objects (called the *page description level* in the figure). Every content stream begins at the page description level, where changes can be made to the graphics state, such as colors and text attributes, as discussed in the following sections.

In the figure, arrows indicate the operators that mark the beginning or end of each type of graphics object. Some operators are identified individually, others by general category. Table 4.1 summarizes these categories for all PDF operators.

TABLE 4.1 Operator categories

CATEGORY	OPERATORS	TABLE	PAGE
General graphics state	w, J, j, M, d, ri, i, gs	4.7	219
Special graphics state	q, Q, cm	4.7	219
Path construction	m, l, c, v, y, h, re	4.9	226
Path painting	S, s, f, F, f*, B, B*, b, b*, n	4.10	230
Clipping paths	W, W*	4.11	235
Text objects	BT, ET	5.4	405
Text state	Tc, Tw, Tz, TL, Tf, Tr, Ts	5.2	398
Text positioning	Td, TD, Tm, T*	5.5	406
Text showing	Tj, TJ, ', "	5.6	407
Type 3 fonts	d0, d1	5.10	423
Color	CS, cs, SC, SCN, sc, scn, G, g, RG, rg, K, k	4.24	287
Shading patterns	sh	4.27	303
Inline images	BI, ID, EI	4.42	352
XObjects	Do	4.37	332
Marked content	MP, DP, BMC, BDC, EMC	10.7	851
Compatibility	BX, EX	3.29	152

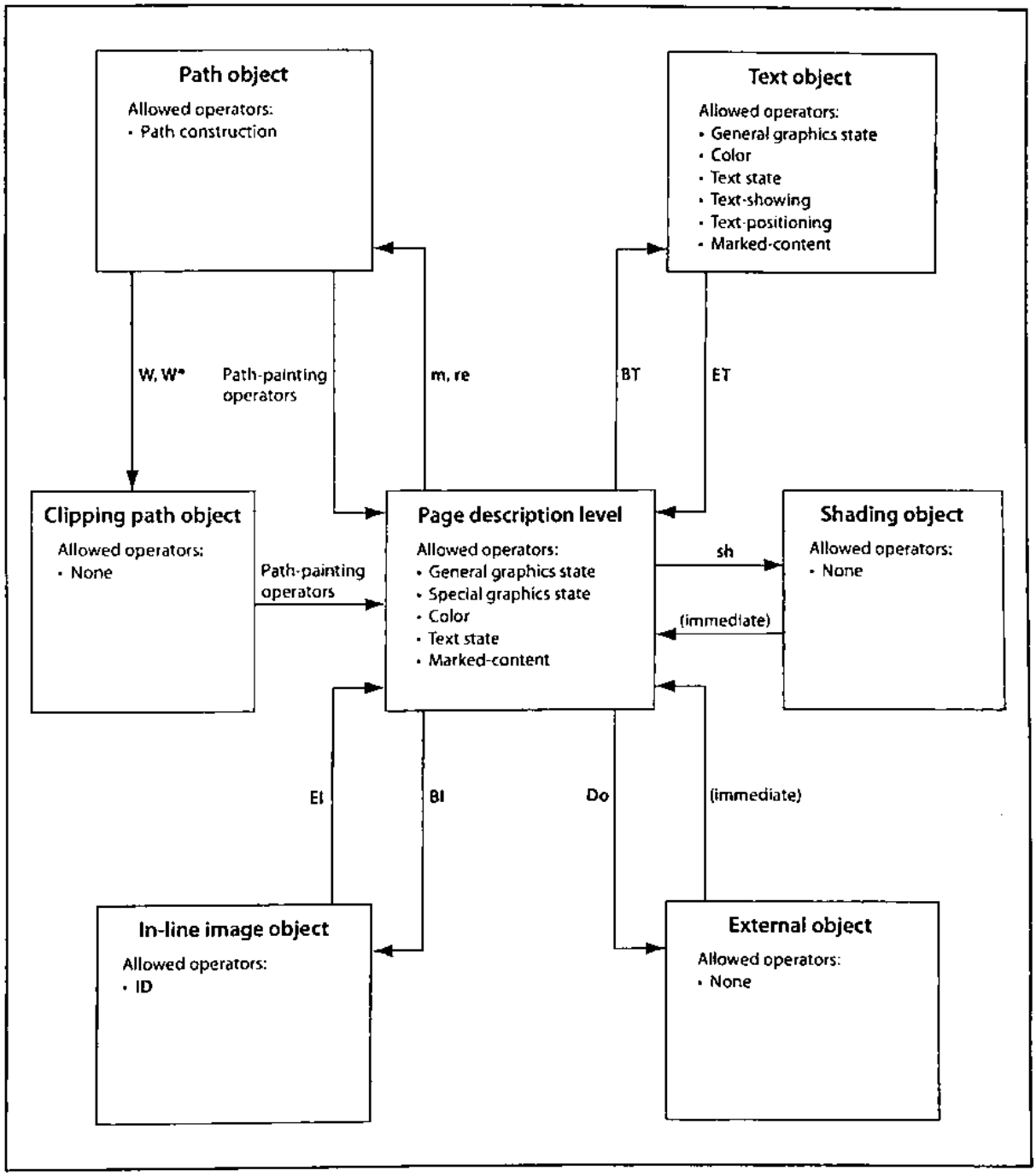


FIGURE 4.1 *Graphics objects*

For example, the path construction operators **m** and **re** signal the beginning of a path object. Inside the path object, additional path construction operators are permitted, as are the clipping path operators **W** and **W***, but not general graphics state operators such as **w** or **J**. A path-painting operator, such as **S** or **f**, ends the path object and returns to the page description level.

Note: *A content stream whose operations violate these rules for describing graphics objects can produce unpredictable behavior, even though it may display and print correctly. Applications that attempt to extract graphics objects for editing or other purposes depend on the objects' being well formed. The rules for graphics objects are also important for the proper interpretation of marked content (see Section 10.5, "Marked Content").*

A graphics object also implicitly includes all graphics state parameters that affect its behavior. For instance, a path object depends on the value of the current color parameter at the moment the path object is defined. The effect is as if this parameter were specified as part of the definition of the path object. However, the operators that are invoked at the page description level to set graphics state parameters are *not* considered to belong to any particular graphics object. Graphics state parameters need to be specified only when they change. A graphics object may depend on parameters that were defined much earlier.

Similarly, the individual character strings within a text object implicitly include the graphics state parameters on which they depend. Most of these parameters may be set inside or outside the text object. The effect is as if they were separately specified for each text string.

The important point is that there is no semantic significance to the exact arrangement of graphics state operators. An application that reads and writes a PDF content stream is not required to preserve this arrangement, but is free to change it to any other arrangement that achieves the same values of the relevant graphics state parameters for each graphics object. An application should not infer any higher-level logical semantics from the arrangement of tokens constituting a graphics object. A separate mechanism, *marked content* (see Section 10.5, "Marked Content"), allows such higher-level information to be explicitly associated with the graphics objects.

4.2 Coordinate Systems

Coordinate systems define the canvas on which all painting occurs. They determine the position, orientation, and size of the text, graphics, and images that appear on a page. This section describes each of the coordinate systems used in PDF, how they are related, and how transformations among them are specified.

Note: The coordinate systems discussed in this section apply to two-dimensional graphics. PDF 1.6 introduces the ability to display 3D artwork, in which objects are described in a three-dimensional coordinate system, as described in Section 9.5.4, "Coordinate Systems for 3D."

4.2.1 Coordinate Spaces

Paths and positions are defined in terms of pairs of *coordinates* on the Cartesian plane. A coordinate pair is a pair of real numbers x and y that locate a point horizontally and vertically within a two-dimensional *coordinate space*. A coordinate space is determined by the following properties with respect to the current page:

- The location of the origin
- The orientation of the x and y axes
- The lengths of the units along each axis

PDF defines several coordinate spaces in which the coordinates specifying graphics objects are interpreted. The following sections describe these spaces and the relationships among them.

Transformations among coordinate spaces are defined by *transformation matrices*, which can specify any linear mapping of two-dimensional coordinates, including translation, scaling, rotation, reflection, and skewing. Transformation matrices are discussed in Sections 4.2.2, "Common Transformations," and 4.2.3, "Transformation Matrices."

Device Space

The contents of a page ultimately appear on a raster output device such as a display or a printer. Such devices vary greatly in the built-in coordinate systems they use to address pixels within their imageable areas. A particular device's coordi-

nate system is called its *device space*. The origin of the device space on different devices can fall in different places on the output page; on displays, the origin can vary depending on the window system. Because the paper or other output medium moves through different printers and imagesetters in different directions, the axes of their device spaces may be oriented differently. For instance, vertical *(y)* coordinates may increase from the top of the page to the bottom on some devices and from bottom to top on others. Finally, different devices have different resolutions; some even have resolutions that differ in the horizontal and vertical directions.

If coordinates in a PDF file were specified in device space, the file would be device-dependent and would appear differently on different devices. For example, images specified in the typical device spaces of a 72-pixel-per-inch display and a 600-dot-per-inch printer would differ in size by more than a factor of 8; an 8-inch line segment on the display would appear less than 1 inch long on the printer. Figure 4.2 shows how the same graphics object, specified in device space, can appear drastically different when rendered on different output devices.

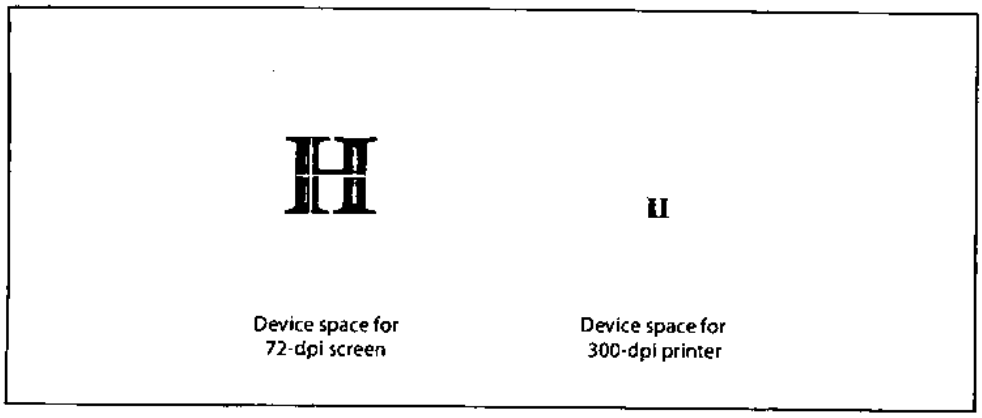


FIGURE 4.2 *Device space*

User Space

To avoid the device-dependent effects of specifying objects in device space, PDF defines a device-independent coordinate system that always bears the same relationship to the current page, regardless of the output device on which printing or displaying occurs. This device-independent coordinate system is called *user space*.

The user space coordinate system is initialized to a default state for each page of a document. The **CropBox** entry in the page dictionary specifies the rectangle of user space corresponding to the visible area of the intended output medium (display window or printed page). The positive *x* axis extends horizontally to the right and the positive *y* axis vertically upward, as in standard mathematical practice (subject to alteration by the **Rotate** entry in the page dictionary). The length of a unit along both the *x* and *y* axes is set by the **UserUnit** entry *(PDF 1.6)* in the page dictionary (see Table 3.27). If that entry is not present or supported, the default value of 1/72 inch is used. This coordinate system is called *default user space.*

Note: In PostScript, the origin of default user space always corresponds to the lower-left corner of the output medium. While this convention is common in PDF documents as well, it is not required; the page dictionary's **CropBox** *entry can specify any rectangle of default user space to be made visible on the medium.*

Note: The default for the size of the unit in default user space (1/72 inch) is approximately the same as a point, *a unit widely used in the printing industry. It is not exactly the same, however; there is no universal definition of a point.*

Conceptually, user space is an infinite plane. Only a small portion of this plane corresponds to the imageable area of the output device: a rectangular region defined by the **CropBox** entry in the page dictionary. The region of default user space that is viewed or printed can be different for each page and is described in Section 10.10.1, "Page Boundaries."

Note: Because coordinates in user space (as in any other coordinate space) may be specified as either integers or real numbers, the unit size in default user space does not constrain positions to any arbitrary grid. The resolution of coordinates in user space is not related in any way to the resolution of pixels in device space.

The transformation from user space to device space is defined by the *current transformation matrix* (CTM), an element of the PDF graphics state (see Section 4.3, "Graphics State"). A PDF consumer application can adjust the CTM for the native resolution of a particular output device, maintaining the device-independence of the PDF page description. Figure 4.3 shows how this allows an object specified in user space to appear the same regardless of the device on which it is rendered.

The default user space provides a consistent, dependable starting place for PDF page descriptions regardless of the output device used. If necessary, a PDF con-

tent stream may modify user space to be more suitable to its needs by applying the *coordinate transformation operator*, **cm** (see Section 4.3.3, "Graphics State Operators"). Thus, what may appear to be absolute coordinates in a content stream are not absolute with respect to the current page because they are expressed in a coordinate system that may slide around and shrink or expand. Coordinate system transformation not only enhances device-independence but is a useful tool in its own right. For example, a content stream originally composed to occupy an entire page can be incorporated without change as an element of another page by shrinking the coordinate system in which it is drawn.

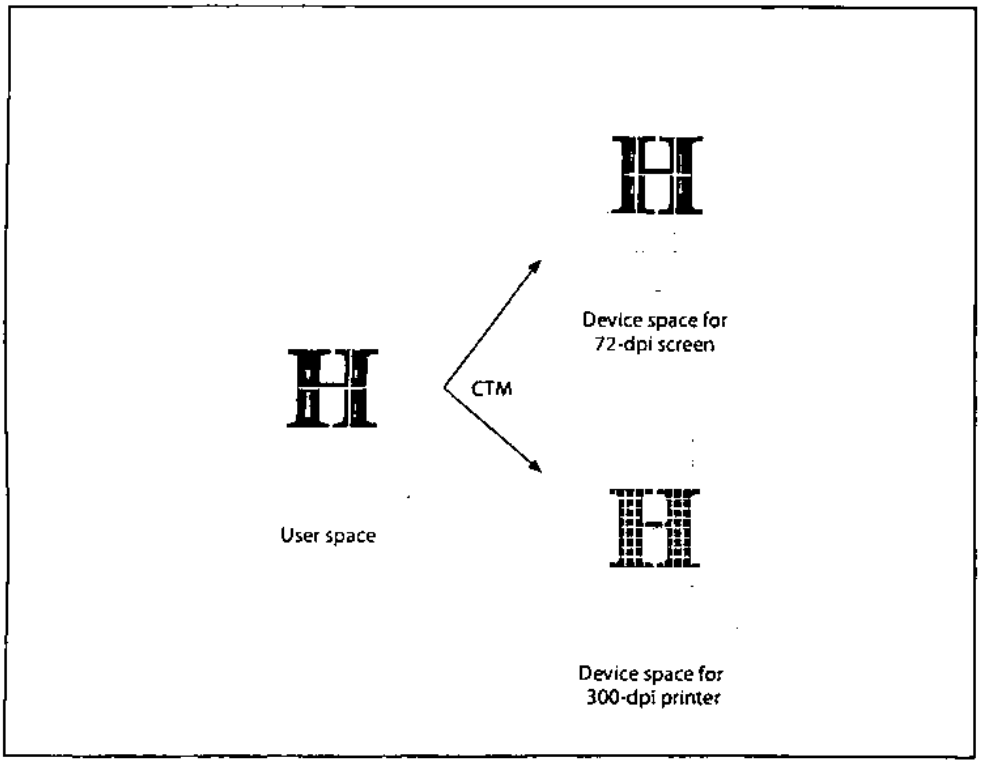


FIGURE 4.3 *User space*

Other Coordinate Spaces

In addition to device space and user space, PDF uses a variety of other coordinate spaces for specialized purposes:

- The coordinates of text are specified in *text space*. The transformation from text space to user space is defined by a *text matrix* in combination with several text-related parameters in the graphics state (see Section 5.3.1, "Text-Positioning Operators").
- Character glyphs in a font are defined in *glyph space* (see Section 5.1.3, "Glyph Positioning and Metrics"). The transformation from glyph space to text space is defined by the *font matrix*. For most types of fonts, this matrix is predefined to map 1000 units of glyph space to 1 unit of text space; for Type 3 fonts, the font matrix is given explicitly in the font dictionary (see Section 5.5.4, "Type 3 Fonts").
- All sampled images are defined in *image space*. The transformation from image space to user space is predefined and cannot be changed. All images are 1 unit wide by 1 unit high in user space, regardless of the number of samples in the image. To be painted, an image is mapped to a region of the page by temporarily altering the CTM.

  Note: In PostScript, unlike PDF, the relationship between image space and user space can be specified explicitly. The fixed transformation prescribed in PDF corresponds to the convention that is recommended for use in PostScript.
- A form XObject (discussed in Section 4.9, "Form XObjects") is a self-contained content stream that can be treated as a graphical element within another content stream. The space in which it is defined is called *form space*. The transformation from form space to user space is specified by a *form matrix* contained in the form XObject.
- PDF 1.2 defines a type of color known as a *pattern*, discussed in Section 4.6, "Patterns." A pattern is defined either by a content stream that is invoked repeatedly to tile an area or by a shading whose color is a function of position. The space in which a pattern is defined is called *pattern space*. The transformation from pattern space to user space is specified by a *pattern matrix* contained in the pattern.
- PDF 1.6 introduces embedded 3D artwork, which is described in three-dimensional coordinates (see Section 9.5.4, "Coordinate Systems for 3D") that are

projected into an annotation's target coordinate system (see Section 9.5.1, "3D Annotations").

Relationships among Coordinate Spaces

Figure 4.4 shows the relationships among the coordinate spaces described above. Each arrow in the figure represents a transformation from one coordinate space to another. PDF allows modifications to many of these transformations.

Because PDF coordinate spaces are defined relative to one another, changes made to one transformation can affect the appearance of objects defined in several coordinate spaces. For example, a change in the CTM, which defines the transformation from user space to device space, affects forms, text, images, and patterns, since they are all upstream from user space.

4.2.2 Common Transformations

A *transformation matrix* specifies the relationship between two coordinate spaces. By modifying a transformation matrix, objects can be scaled, rotated, translated, or transformed in other ways.

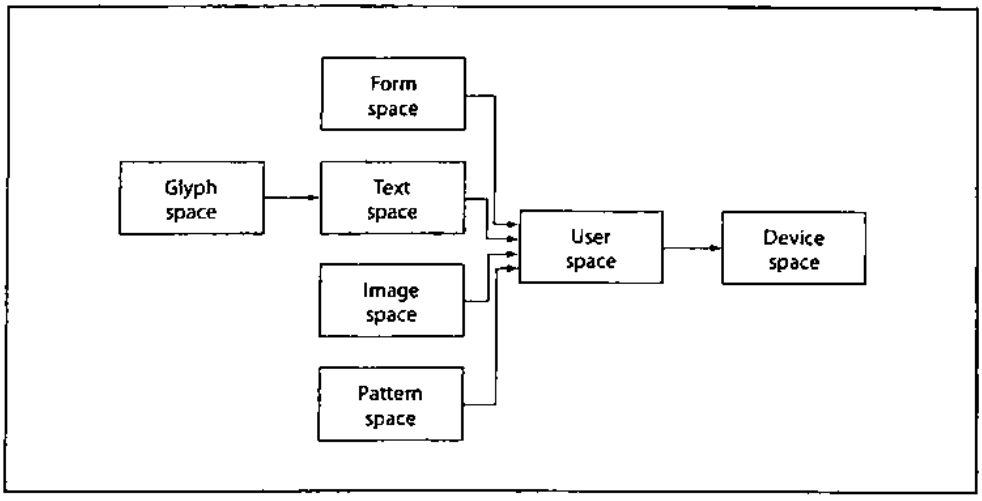


FIGURE 4.4 *Relationships among coordinate systems*

A transformation matrix in PDF is specified by six numbers, usually in the form of an array containing six elements. In its most general form, this array is denoted [*a b c d e f*]; it can represent any linear transformation from one coordinate system to another. This section lists the arrays that specify the most common transformations; Section 4.2.3, "Transformation Matrices," discusses more mathematical details of transformations, including information on specifying transformations that are combinations of those listed here:

- Translations are specified as [1 0 0 1 t_x t_y], where t_x and t_y are the distances to translate the origin of the coordinate system in the horizontal and vertical dimensions, respectively.
- Scaling is obtained by [s_x 0 0 s_y 0 0]. This scales the coordinates so that 1 unit in the horizontal and vertical dimensions of the new coordinate system is the same size as s_x and s_y units, respectively, in the previous coordinate system.
- Rotations are produced by [cos θ sin θ −sin θ cos θ 0 0], which has the effect of rotating the coordinate system axes by an angle θ counterclockwise.
- Skew is specified by [1 tan α tan β 1 0 0], which skews the x axis by an angle α and the y axis by an angle β.

Figure 4.5 shows examples of each transformation. The directions of translation, rotation, and skew shown in the figure correspond to positive values of the array elements.

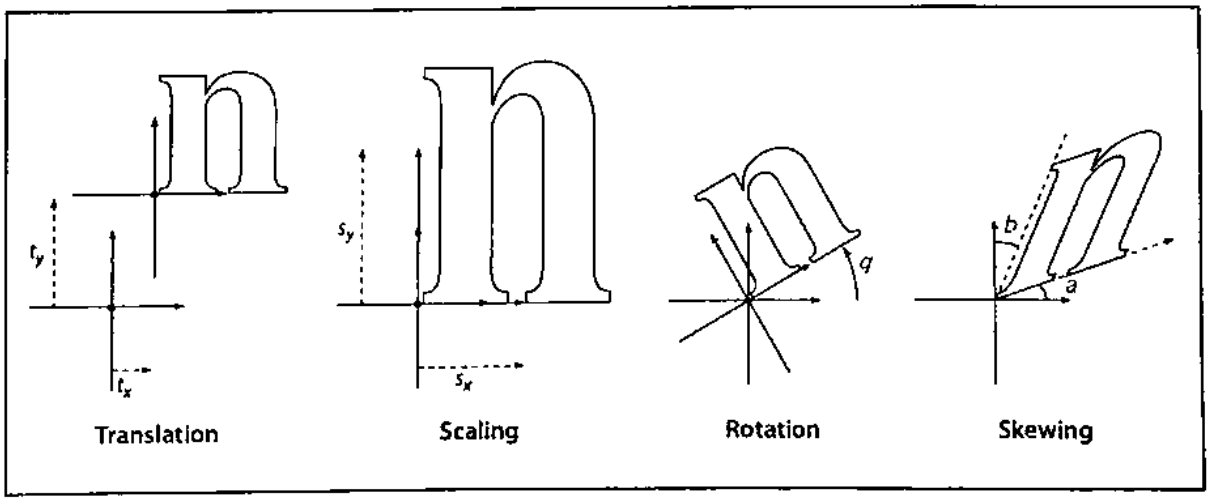


FIGURE 4.5 *Effects of coordinate transformations*

If several transformations are combined, the order in which they are applied is significant. For example, first scaling and then translating the x axis is not the same as first translating and then scaling it. In general, to obtain the expected results, transformations should be done in the following order:

1. Translate
2. Rotate
3. Scale or skew

Figure 4.6 shows the effect of the order in which transformations are applied. The figure shows two sequences of transformations applied to a coordinate system. After each successive transformation, an outline of the letter n is drawn.

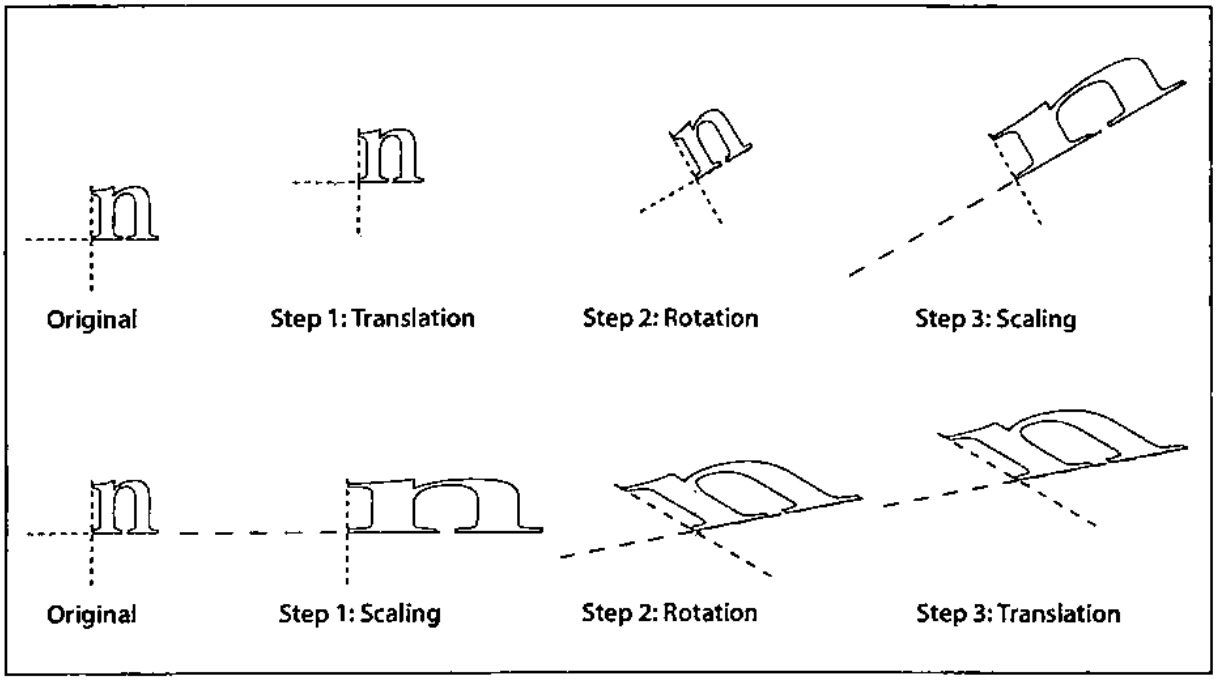


FIGURE 4.6 *Effect of transformation order*

The following transformations are shown in the figure:

- A translation of 10 units in the x direction and 20 units in the y direction
- A rotation of 30 degrees
- A scaling by a factor of 3 in the x direction

In the figure, the axes are shown with a dash pattern having a 2-unit dash and a 2-unit gap. In addition, the original (untransformed) axes are shown in a lighter color for reference. Notice that the scale-rotate-translate ordering results in a distortion of the coordinate system, leaving the x and y axes no longer perpendicular; the recommended translate-rotate-scale ordering results in no distortion.

4.2.3 Transformation Matrices

This section discusses the mathematics of transformation matrices. It is not necessary to read this section to use the transformations described previously; the information is presented for the benefit of readers who want to gain a deeper understanding of the theoretical basis of coordinate transformations.

To understand the mathematics of coordinate transformations in PDF, it is vital to remember two points:

- *Transformations alter coordinate systems, not graphics objects.* All objects painted before a transformation is applied are unaffected by the transformation. Objects painted after the transformation is applied are interpreted in the transformed coordinate system.
- *Transformation matrices specify the transformation from the new (transformed) coordinate system to the original (untransformed) coordinate system.* All coordinates used after the transformation are expressed in the transformed coordinate system. PDF applies the transformation matrix to find the equivalent coordinates in the untransformed coordinate system.

Note: *Many computer graphics textbooks consider transformations of graphics objects rather than of coordinate systems. Although either approach is correct and self-consistent, some details of the calculations differ depending on which point of view is taken.*

PDF represents coordinates in a two-dimensional space. The point (x, y) in such a space can be expressed in vector form as $[x\ y\ 1]$. The constant third element of this vector (1) is needed so that the vector can be used with 3-by-3 matrices in the calculations described below.

The transformation between two coordinate systems is represented by a 3-by-3 transformation matrix written as follows:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Because a transformation matrix has only six elements that can be changed, it is usually specified in PDF as the six-element array $[a\ b\ c\ d\ e\ f]$.

Coordinate transformations are expressed as matrix multiplications:

$$[x'\ y'\ 1] = [x\ y\ 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Because PDF transformation matrices specify the conversion from the transformed coordinate system to the original (untransformed) coordinate system, x' and y' in this equation are the coordinates in the untransformed coordinate system, and x and y are the coordinates in the transformed system. The multiplication is carried out as follows:

$$x' = a \times x + c \times y + e$$
$$y' = b \times x + d \times y + f$$

If a series of transformations is carried out, the matrices representing each of the individual transformations can be multiplied together to produce a single equivalent matrix representing the composite transformation.

Matrix multiplication is not commutative—the order in which matrices are multiplied is significant. Consider a sequence of two transformations: a scaling transformation applied to the user space coordinate system, followed by a conversion from the resulting scaled user space to device space. Let M_S be the matrix specifying the scaling and M_C the current transformation matrix, which transforms user

space to device space. Recalling that coordinates are always specified in the transformed space, the correct order of transformations must first convert the scaled coordinates to default user space and then the default user space coordinates to device space. This can be expressed as

$$X_D = X_U \times M_C = (X_S \times M_S) \times M_C = X_S \times (M_S \times M_C)$$

where

X_D denotes the coordinates in device space

X_U denotes the coordinates in default user space

X_S denotes the coordinates in scaled user space

This shows that when a new transformation is concatenated with an existing one, the matrix representing it must be multiplied *before* (*premultiplied* with) the existing transformation matrix.

This result is true in general for PDF: when a sequence of transformations is carried out, the matrix representing the combined transformation *(M′)* is calculated by premultiplying the matrix representing the additional transformation (M_T) with the one representing all previously existing transformations *(M)*:

$$M' = M_T \times M$$

Note: *When rendering graphics objects, it is sometimes necessary for an application to perform the inverse of a transformation—that is, to find the user space coordinates that correspond to a given pair of device space coordinates. Not all transformations are invertible, however. For example, if a matrix contains* a, b, c, *and* d *elements that are all zero, all user coordinates map to the same device coordinates and there is no unique inverse transformation. Such noninvertible transformations are not very useful and generally arise from unintended operations, such as scaling by 0. Use of a noninvertible matrix when painting graphics objects can result in unpredictable behavior.*

4.3 Graphics State

A PDF consumer application maintains an internal data structure called the *graphics state* that holds current graphics control parameters. These parameters define the global framework within which the graphics operators execute. For example, the **f** (fill) operator implicitly uses the *current color* parameter, and the **S** (stroke) operator additionally uses the *current line width* parameter from the graphics state.

The graphics state is initialized at the beginning of each page with the values specified in Tables 4.2 and 4.3. Table 4.2 lists those graphics state parameters that are device-independent and are appropriate to specify in page descriptions. The parameters listed in Table 4.3 control details of the rendering (scan conversion) process and are device-dependent; a page description that is intended to be device-independent should not modify these parameters.

TABLE 4.2 Device-Independent graphics state parameters

PARAMETER	TYPE	VALUE
CTM	array	The *current transformation matrix*, which maps positions from user coordinates to device coordinates (see Section 4.2, "Coordinate Systems"). This matrix is modified by each application of the coordinate transformation operator, **cm**. Initial value: a matrix that transforms default user coordinates to device coordinates.
clipping path	(internal)	The *current clipping path*, which defines the boundary against which all output is to be cropped (see Section 4.4.3, "Clipping Path Operators"). Initial value: the boundary of the entire imageable portion of the output page.
color space	name or array	The *current color space* in which color values are to be interpreted (see Section 4.5, "Color Spaces"). There are two separate color space parameters: one for stroking and one for all other painting operations. Initial value: **DeviceGray**.
color	(various)	The *current color* to be used during painting operations (see Section 4.5, "Color Spaces"). The type and interpretation of this parameter depend on the current color space; for most color spaces, a color value consists of one to four numbers. There are two separate color parameters: one for stroking and one for all other painting operations. Initial value: black.

PARAMETER	TYPE	VALUE
text state	(various)	A set of nine graphics state parameters that pertain only to the painting of text. These include parameters that select the font, scale the glyphs to an appropriate size, and accomplish other effects. The text state parameters are described in Section 5.2, "Text State Parameters and Operators."
line width	number	The thickness, in user space units, of paths to be stroked (see "Line Width" on page 215). Initial value: 1.0.
line cap	integer	A code specifying the shape of the endpoints for any open path that is stroked (see "Line Cap Style" on page 216). Initial value: 0, for square butt caps.
line join	integer	A code specifying the shape of joints between connected segments of a stroked path (see "Line Join Style" on page 216). Initial value: 0, for mitered joins.
miter limit	number	The maximum length of mitered line joins for stroked paths (see "Miter Limit" on page 217). This parameter limits the length of "spikes" produced when line segments join at sharp angles. Initial value: 10.0, for a miter cutoff below approximately 11.5 degrees.
dash pattern	array and number	A description of the dash pattern to be used when paths are stroked (see "Line Dash Pattern" on page 217). Initial value: a solid line.
rendering intent	name	The *rendering intent* to be used when converting CIE-based colors to device colors (see "Rendering Intents" on page 260). Initial value: **RelativeColorimetric**.
stroke adjustment	boolean	*(PDF 1.2)* A flag specifying whether to compensate for possible rasterization effects when stroking a path with a line width that is small relative to the pixel resolution of the output device (see Section 6.5.4, "Automatic Stroke Adjustment"). Note that this is considered a device-independent parameter, even though the details of its effects are device-dependent. Initial value: **false**.
blend mode	name or array	*(PDF 1.4)* The *current blend mode* to be used in the transparent imaging model (see Sections 7.2.4, "Blend Mode," and 7.5.2, "Specifying Blending Color Space and Blend Mode"). This parameter is implicitly reset to its initial value at the beginning of execution of a transparency group XObject (see Section 7.5.5, "Transparency Group XObjects"). Initial value: **Normal**.

PARAMETER	TYPE	VALUE
soft mask	dictionary or name	*(PDF 1.4)* A *soft-mask dictionary* (see "Soft-Mask Dictionaries" on page 552) specifying the mask shape or mask opacity values to be used in the transparent imaging model (see "Source Shape and Opacity" on page 526 and "Mask Shape and Opacity" on page 550), or the name **None** if no such mask is specified. This parameter is implicitly reset to its initial value at the beginning of execution of a transparency group XObject (see Section 7.5.5, "Transparency Group XObjects"). Initial value: **None**.
alpha constant	number	*(PDF 1.4)* The constant shape or constant opacity value to be used in the transparent imaging model (see "Source Shape and Opacity" on page 526 and "Constant Shape and Opacity" on page 551). There are two separate alpha constant parameters: one for stroking and one for all other painting operations. This parameter is implicitly reset to its initial value at the beginning of execution of a transparency group XObject (see Section 7.5.5, "Transparency Group XObjects"). Initial value: 1.0.
alpha source	boolean	*(PDF 1.4)* A flag specifying whether the current soft mask and alpha constant parameters are to be interpreted as shape values (**true**) or opacity values (**false**). This flag also governs the interpretation of the **SMask** entry, if any, in an image dictionary (see Section 4.8.4, "Image Dictionaries"). Initial value: **false**.

TABLE 4.3 Device-dependent graphics state parameters

PARAMETER	TYPE	VALUE
overprint	boolean	*(PDF 1.2)* A flag specifying (on output devices that support the overprint control feature) whether painting in one set of colorants should cause the corresponding areas of other colorants to be erased (**false**) or left unchanged (**true**); see Section 4.5.6, "Overprint Control." In PDF 1.3, there are two separate overprint parameters: one for stroking and one for all other painting operations. Initial value: **false**.
overprint mode	number	*(PDF 1.3)* A code specifying whether a color component value of 0 in a **DeviceCMYK** color space should erase that component (0) or leave it unchanged (1) when overprinting (see Section 4.5.6, "Overprint Control"). Initial value: 0.

PARAMETER	TYPE	VALUE
black generation	function or name	*(PDF 1.2)* A function that calculates the level of the black color component to use when converting *RGB* colors to *CMYK* (see Section 6.2.3, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: installation-dependent.
undercolor removal	function or name	*(PDF 1.2)* A function that calculates the reduction in the levels of the cyan, magenta, and yellow color components to compensate for the amount of black added by black generation (see Section 6.2.3, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: installation-dependent.
transfer	function, array, or name	*(PDF 1.2)* A function that adjusts device gray or color component levels to compensate for nonlinear response in a particular output device (see Section 6.3, "Transfer Functions"). Initial value: installation-dependent.
halftone	dictionary, stream, or name	*(PDF 1.2)* A halftone screen for gray and color rendering, specified as a halftone dictionary or stream (see Section 6.4, "Halftones"). Initial value: installation-dependent.
flatness	number	The precision with which curves are to be rendered on the output device (see Section 6.5.1, "Flatness Tolerance"). The value of this parameter gives the maximum error tolerance, measured in output device pixels; smaller numbers give smoother curves at the expense of more computation and memory use. Initial value: 1.0.
smoothness	number	*(PDF 1.3)* The precision with which color gradients are to be rendered on the output device (see Section 6.5.2, "Smoothness Tolerance"). The value of this parameter gives the maximum error tolerance, expressed as a fraction of the range of each color component; smaller numbers give smoother color transitions at the expense of more computation and memory use. Initial value: installation-dependent.

Some graphics state parameters are set with specific PDF operators, some are set by including a particular entry in a *graphics state parameter dictionary*, and some can be specified either way. The current line width, for example, can be set either with the **w** operator or (in PDF 1.3) with the **LW** entry in a graphics state parameter dictionary, whereas the current color is set only with specific operators, and the current halftone is set only with a graphics state parameter dictionary. It is expected that all future graphics state parameters will be specified with new entries in the graphics state parameter dictionary rather than with new operators.

In general, the operators that set graphics state parameters simply store them unchanged for later use by the painting operators. However, some parameters have special properties or behavior:

- Most parameters must be of the correct type or have values that fall within a certain range.
- Parameters that are numeric values, such as the current color, line width, and miter limit, are forced into valid range, if necessary. However, they are *not* adjusted to reflect capabilities of the raster output device, such as resolution or number of distinguishable colors. Painting operators perform such adjustments, but the adjusted values are not stored back into the graphics state.
- Paths are internal objects that are not directly represented in PDF.

Note: As indicated in Tables 4.2 and 4.3, some of the parameters—color space, color, and overprint—have two values, one used for stroking (of paths and text objects) and one for all other painting operations. The two parameter values can be set independently, allowing for operations such as combined filling and stroking of the same path with different colors. Except where noted, a term such as current color *should be interpreted to refer to whichever color parameter applies to the operation being performed. When necessary, the individual color parameters are distinguished explicitly as the* stroking color *and the* nonstroking color.

4.3.1 Graphics State Stack

A well-structured PDF document typically contains many graphical elements that are essentially independent of each other and sometimes nested to multiple levels. The *graphics state stack* allows these elements to make local changes to the graphics state without disturbing the graphics state of the surrounding environment. The stack is a LIFO (last in, first out) data structure in which the contents of the graphics state can be saved and later restored using the following operators:

- The q operator pushes a copy of the entire graphics state onto the stack.
- The Q operator restores the entire graphics state to its former value by popping it from the stack.

These operators can be used to encapsulate a graphical element so that it can modify parameters of the graphics state and later restore them to their previous values. Occurrences of the **q** and **Q** operators must be balanced within a given content stream (or within the sequence of streams specified in a page dictionary's **Contents** array).

4.3.2 Details of Graphics State Parameters

This section gives details of several of the device-independent graphics state parameters listed in Table 4.2.

Line Width

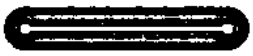
The *line width* parameter specifies the thickness of the line used to stroke a path. It is a non-negative number expressed in user space units; stroking a path entails painting all points whose perpendicular distance from the path in user space is less than or equal to half the line width. The effect produced in device space depends on the current transformation matrix (CTM) in effect at the time the path is stroked. If the CTM specifies scaling by different factors in the horizontal and vertical dimensions, the thickness of stroked lines in device space will vary according to their orientation. The actual line width achieved can differ from the requested width by as much as 2 device pixels, depending on the positions of lines with respect to the pixel grid. Automatic stroke adjustment can be used to ensure uniform line width; see Section 6.5.4, "Automatic Stroke Adjustment."

A line width of 0 denotes the thinnest line that can be rendered at device resolution: 1 device pixel wide. However, some devices cannot reproduce 1-pixel lines, and on high-resolution devices, they are nearly invisible. Since the results of rendering such zero-width lines are device-dependent, their use is not recommended.

Line Cap Style

The *line cap style* specifies the shape to be used at the ends of open subpaths (and dashes, if any) when they are stroked. Table 4.4 shows the possible values.

TABLE 4.4 Line cap styles

STYLE	APPEARANCE	DESCRIPTION
0		*Butt cap.* The stroke is squared off at the endpoint of the path. There is no projection beyond the end of the path.
1		*Round cap.* A semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.
2		*Projecting square cap.* The stroke continues beyond the endpoint of the path for a distance equal to half the line width and is squared off.

Line Join Style

The *line join style* specifies the shape to be used at the corners of paths that are stroked. Table 4.5 shows the possible values. Join styles are significant only at points where consecutive segments of a path connect at an angle; segments that meet or intersect fortuitously receive no special treatment.

TABLE 4.5 Line join styles

STYLE	APPEARANCE	DESCRIPTION
0		*Miter join.* The outer edges of the strokes for the two segments are extended until they meet at an angle, as in a picture frame. If the segments meet at too sharp an angle (as defined by the miter limit parameter—see "Miter Limit," above), a bevel join is used instead.
1		*Round join.* An arc of a circle with a diameter equal to the line width is drawn around the point where the two segments meet, connecting the outer edges of the strokes for the two segments. This pieslice-shaped figure is filled in, producing a rounded corner.
2		*Bevel join.* The two segments are finished with butt caps (see "Line Cap Style" on page 216) and the resulting notch beyond the ends of the segments is filled with a triangle.

Note: The definition of round join was changed in PDF 1.5. In rare cases, the implementation of the previous specification could produce unexpected results.

Miter Limit

When two line segments meet at a sharp angle and mitered joins have been specified as the line join style, it is possible for the miter to extend far beyond the thickness of the line stroking the path. The *miter limit* imposes a maximum on the ratio of the miter length to the line width (see Figure 4.7). When the limit is exceeded, the join is converted from a miter to a bevel.

The ratio of miter length to line width is directly related to the angle φ between the segments in user space by the following formula:

$$\frac{miterLength}{lineWidth} = \frac{1}{\sin\left(\frac{\varphi}{2}\right)}$$

For example, a miter limit of 1.414 converts miters to bevels for φ less than 90 degrees, a limit of 2.0 converts them for φ less than 60 degrees, and a limit of 10.0 converts them for φ less than approximately 11.5 degrees.

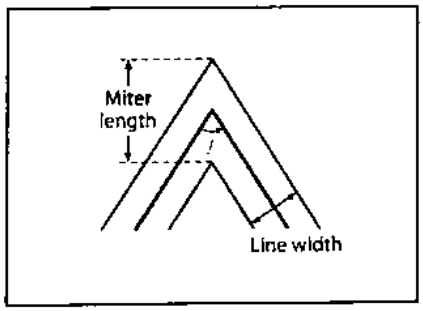


FIGURE 4.7 *Miter length*

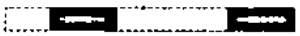
Line Dash Pattern

The *line dash pattern* controls the pattern of dashes and gaps used to stroke paths. It is specified by a *dash array* and a *dash phase*. The dash array's elements are numbers that specify the lengths of alternating dashes and gaps; the numbers must be nonnegative and not all zero. The dash phase specifies the distance into

the dash pattern at which to start the dash. The elements of both the dash array and the dash phase are expressed in user space units.

Before beginning to stroke a path, the dash array is cycled through, adding up the lengths of dashes and gaps. When the accumulated length equals the value specified by the dash phase, stroking of the path begins, and the dash array is used cyclically from that point onward. Table 4.6 shows examples of line dash patterns. As can be seen from the table, an empty dash array and zero phase can be used to restore the dash pattern to a solid line.

TABLE 4.6 Examples of line dash patterns

DASH ARRAY AND PHASE	APPEARANCE	DESCRIPTION
[] 0		No dash; solid, unbroken lines
[3] 0		3 units on, 3 units off, ...
[2] 1		1 on, 2 off, 2 on, 2 off, ...
[2 1] 0		2 on, 1 off, 2 on, 1 off, ...
[3 5] 6		2 off, 3 on, 5 off, 3 on, 5 off, ...
[2 3] 11		1 on, 3 off, 2 on, 3 off, 2 on, ...

Dashed lines wrap around curves and corners just as solid stroked lines do. The ends of each dash are treated with the current line cap style, and corners within dashes are treated with the current line join style. A stroking operation takes no measures to coordinate the dash pattern with features of the path; it simply dispenses dashes and gaps along the path in the pattern defined by the dash array.

When a path consisting of several subpaths is stroked, each subpath is treated independently—that is, the dash pattern is restarted and the dash phase is reapplied to it at the beginning of each subpath.

4.3.3 Graphics State Operators

Table 4.7 shows the operators that set the values of parameters in the graphics state. (See also the color operators listed in Table 4.24 and the text state operators in Table 5.2 on page 398.)

TABLE 4.7 Graphics state operators

OPERANDS	OPERATOR	DESCRIPTION
—	**q**	Save the current graphics state on the graphics state stack (see "Graphics State Stack" on page 214).
—	**Q**	Restore the graphics state by removing the most recently saved state from the stack and making it the current state (see "Graphics State Stack" on page 214).
a b c d e f	**cm**	Modify the current transformation matrix (CTM) by concatenating the specified matrix (see Section 4.2.1, "Coordinate Spaces"). Although the operands specify a matrix, they are written as six separate numbers, not as an array.
lineWidth	**w**	Set the line width in the graphics state (see "Line Width" on page 215).
lineCap	**J**	Set the line cap style in the graphics state (see "Line Cap Style" on page 216).
lineJoin	**j**	Set the line join style in the graphics state (see "Line Join Style" on page 216).
miterLimit	**M**	Set the miter limit in the graphics state (see "Miter Limit" on page 217).
dashArray dashPhase	**d**	Set the line dash pattern in the graphics state (see "Line Dash Pattern" on page 217).
intent	**ri**	*(PDF 1.1)* Set the color rendering intent in the graphics state (see "Rendering Intents" on page 260).
flatness	**i**	Set the flatness tolerance in the graphics state (see Section 6.5.1, "Flatness Tolerance"). *flatness* is a number in the range 0 to 100; a value of 0 specifies the output device's default flatness tolerance.
dictName	**gs**	*(PDF 1.2)* Set the specified parameters in the graphics state. *dictName* is the name of a graphics state parameter dictionary in the **ExtGState** subdictionary of the current resource dictionary (see the next section).

4.3.4 Graphics State Parameter Dictionaries

While some parameters in the graphics state can be set with individual operators, as shown in Table 4.7, others cannot. The latter can only be set with the generic graphics state operator **gs** *(PDF 1.2)*. The operand supplied to this operator is the

name of a *graphics state parameter dictionary* whose contents specify the values of one or more graphics state parameters. This name is looked up in the **ExtGState** subdictionary of the current resource dictionary. (The name **ExtGState**, for *extended graphics state*, is a vestige of earlier versions of PDF.)

Note: The graphics state parameter dictionary is also used by type 2 patterns, which do not have a content stream in which the graphics state operators could be invoked (see Section 4.6.3, "Shading Patterns").

Each entry in the parameter dictionary specifies the value of an individual graphics state parameter, as shown in Table 4.8. All entries need not be present for every invocation of the **gs** operator; the supplied parameter dictionary may include any combination of parameter entries. The results of **gs** are cumulative; parameter values established in previous invocations persist until explicitly overridden. Note that some parameters appear in both Tables 4.7 and 4.8; these parameters can be set either with individual graphics state operators or with **gs**. It is expected that any future extensions to the graphics state will be implemented by adding new entries to the graphics state parameter dictionary rather than by introducing new graphics state operators.

TABLE 4.8 Entries in a graphics state parameter dictionary

KEY	TYPE	DESCRIPTION
Type	name	*(Optional)* The type of PDF object that this dictionary describes; must be **ExtGState** for a graphics state parameter dictionary.
LW	number	*(Optional; PDF 1.3)* The line width (see "Line Width" on page 215).
LC	integer	*(Optional; PDF 1.3)* The line cap style (see "Line Cap Style" on page 216).
LJ	integer	*(Optional; PDF 1.3)* The line join style (see "Line Join Style" on page 216).
ML	number	*(Optional; PDF 1.3)* The miter limit (see "Miter Limit" on page 217).
D	array	*(Optional; PDF 1.3)* The line dash pattern, expressed as an array of the form [*dashArray dashPhase*], where *dashArray* is itself an array and *dashPhase* is an integer (see "Line Dash Pattern" on page 217).
RI	name	*(Optional; PDF 1.3)* The name of the rendering intent (see "Rendering Intents" on page 260).

KEY	TYPE	DESCRIPTION
OP	boolean	*(Optional)* A flag specifying whether to apply overprint (see Section 4.5.6, "Overprint Control"). In PDF 1.2 and earlier, there is a single overprint parameter that applies to all painting operations. Beginning with PDF 1.3, there are two separate overprint parameters: one for stroking and one for all other painting operations. Specifying an **OP** entry sets both parameters unless there is also an **op** entry in the same graphics state parameter dictionary, in which case the **OP** entry sets only the overprint parameter for stroking.
op	boolean	*(Optional; PDF 1.3)* A flag specifying whether to apply overprint (see Section 4.5.6, "Overprint Control") for painting operations other than stroking. If this entry is absent, the **OP** entry, if any, sets this parameter.
OPM	integer	*(Optional; PDF 1.3)* The overprint mode (see Section 4.5.6, "Overprint Control").
Font	array	*(Optional; PDF 1.3)* An array of the form [*font size*], where *font* is an indirect reference to a font dictionary and *size* is a number expressed in text space units. These two objects correspond to the operands of the **Tf** operator (see Section 5.2, "Text State Parameters and Operators"); however, the first operand is an indirect object reference instead of a resource name.
BG	function	*(Optional)* The black-generation function, which maps the interval [0.0 1.0] to the interval [0.0 1.0] (see Section 6.2.3, "Conversion from DeviceRGB to DeviceCMYK").
BG2	function or name	*(Optional; PDF 1.3)* Same as **BG** except that the value may also be the name Default, denoting the black-generation function that was in effect at the start of the page. If both **BG** and **BG2** are present in the same graphics state parameter dictionary, **BG2** takes precedence.
UCR	function	*(Optional)* The undercolor-removal function, which maps the interval [0.0 1.0] to the interval [−1.0 1.0] (see Section 6.2.3, "Conversion from DeviceRGB to DeviceCMYK").
UCR2	function or name	*(Optional; PDF 1.3)* Same as **UCR** except that the value may also be the name Default, denoting the undercolor-removal function that was in effect at the start of the page. If both **UCR** and **UCR2** are present in the same graphics state parameter dictionary, **UCR2** takes precedence.

KEY	TYPE	DESCRIPTION
TR	function, array, or name	*(Optional)* The transfer function, which maps the interval [0.0 1.0] to the interval [0.0 1.0] (see Section 6.3, "Transfer Functions"). The value is either a single function (which applies to all process colorants) or an array of four functions (which apply to the process colorants individually). The name **Identity** may be used to represent the identity function.
TR2	function, array, or name	*(Optional; PDF 1.3)* Same as **TR** except that the value may also be the name Default, denoting the transfer function that was in effect at the start of the page. If both **TR** and **TR2** are present in the same graphics state parameter dictionary, **TR2** takes precedence.
HT	dictionary, stream, or name	*(Optional)* The halftone dictionary or stream (see Section 6.4, "Halftones") or the name Default, denoting the halftone that was in effect at the start of the page.
FL	number	*(Optional; PDF 1.3)* The flatness tolerance (see Section 6.5.1, "Flatness Tolerance").
SM	number	*(Optional; PDF 1.3)* The smoothness tolerance (see Section 6.5.2, "Smoothness Tolerance").
SA	boolean	*(Optional)* A flag specifying whether to apply automatic stroke adjustment (see Section 6.5.4, "Automatic Stroke Adjustment").
BM	name or array	*(Optional; PDF 1.4)* The current blend mode to be used in the transparent imaging model (see Sections 7.2.4, "Blend Mode," and 7.5.2, "Specifying Blending Color Space and Blend Mode").
SMask	dictionary or name	*(Optional; PDF 1.4)* The current soft mask, specifying the mask shape or mask opacity values to be used in the transparent imaging model (see "Source Shape and Opacity" on page 526 and "Mask Shape and Opacity" on page 550). ***Note:** Although the current soft mask is sometimes referred to as a "soft clip," altering it with the **gs** operator completely replaces the old value with the new one, rather than intersecting the two as is done with the current clipping path parameter (see Section 4.4.3, "Clipping Path Operators").*
CA	number	*(Optional; PDF 1.4)* The current stroking alpha constant, specifying the constant shape or constant opacity value to be used for stroking operations in the transparent imaging model (see "Source Shape and Opacity" on page 526 and "Constant Shape and Opacity" on page 551).
ca	number	*(Optional; PDF 1.4)* Same as **CA**, but for nonstroking operations.

KEY	TYPE	DESCRIPTION
AIS	boolean	*(Optional; PDF 1.4)* The alpha source flag ("alpha is shape"), specifying whether the current soft mask and alpha constant are to be interpreted as shape values (**true**) or opacity values (**false**).
TK	boolean	*(Optional; PDF 1.4)* The text knockout flag, which determines the behavior of overlapping glyphs within a text object in the transparent imaging model (see Section 5.2.7, "Text Knockout").

Example 4.1 shows two graphics state parameter dictionaries. In the first, automatic stroke adjustment is turned on, and the dictionary includes a transfer function that inverts its value, $f(x) = 1 - x$. In the second, overprint is turned off, and the dictionary includes a parabolic transfer function, $f(x) = (2x - 1)^2$, with a sample of 21 values. The domain of the transfer function, [0.0 1.0], is mapped to [0 20], and the range of the sample values, [0 255], is mapped to the range of the transfer function, [0.0 1.0].

Example 4.1

```
10 0 obj                                    % Page object
   << /Type /Page
      /Parent 5 0 R
      /Resources 20 0 R
      /Contents 40 0 R
   >>
endobj

20 0 obj                                    % Resource dictionary for page
   << /ProcSet [/PDF /Text]
      /Font << /F1 25 0 R >>
      /ExtGState << /GS1 30 0 R
                    /GS2 35 0 R
                 >>
   >>
endobj

30 0 obj                                    % First graphics state parameter dictionary
   << /Type /ExtGState
      /SA true
      /TR 31 0 R
   >>
endobj
```

```
31 0 obj                                % First transfer function
   << /FunctionType 0
      /Domain [0.0 1.0]
      /Range [0.0 1.0]
      /Size 2
      /BitsPerSample 8
      /Length 7
      /Filter /ASCIIHexDecode
   >>
stream
01 00 >
endstream
endobj

35 0 obj                                % Second graphics state parameter dictionary
   << /Type /ExtGState
      /OP false
      /TR 36 0 R
   >>
endobj

36 0 obj                                % Second transfer function
   << /FunctionType 0
      /Domain [0.0 1.0]
      /Range [0.0 1.0]
      /Size 21
      /BitsPerSample 8
      /Length 63
      /Filter /ASCIIHexDecode
   >>
stream
FF CE A3 7C 5B 3F 28 16 0A 02 00 02 0A 16 28 3F 5B 7C A3 CE FF >
endstream
endobj
```

4.4 Path Construction and Painting

Paths define shapes, trajectories, and regions of all sorts. They are used to draw lines, define the shapes of filled areas, and specify boundaries for clipping other graphics. The graphics state includes a *current clipping path* that defines the clipping boundary for the current page. At the beginning of each page, the clipping path is initialized to include the entire page.

A path is composed of straight and curved line segments, which may connect to one another or may be disconnected. A pair of segments are said to *connect* only if they are defined consecutively, with the second segment starting where the first one ends. Thus, the order in which the segments of a path are defined is significant. Nonconsecutive segments that meet or intersect fortuitously are not considered to connect.

A path is made up of one or more disconnected *subpaths*, each comprising a sequence of connected segments. The topology of the path is unrestricted: it may be concave or convex, may contain multiple subpaths representing disjoint areas, and may intersect itself in arbitrary ways. The **h** operator explicitly connects the end of a subpath back to its starting point; such a subpath is said to be *closed*. A subpath that has not been explicitly closed is *open*.

As discussed in Section 4.1, "Graphics Objects," a path object is defined by a sequence of operators to construct the path, followed by one or more operators to paint the path or to use it as a clipping boundary. PDF path operators fall into three categories:

- *Path construction operators* (Section 4.4.1) define the geometry of a path. A path is constructed by sequentially applying one or more of these operators.
- *Path-painting operators* (Section 4.4.2) end a path object, usually causing the object to be painted on the current page in any of a variety of ways.
- *Clipping path operators* (Section 4.4.3), invoked immediately before a path-painting operator, cause the path object also to be used for clipping of subsequent graphics objects.

4.4.1 Path Construction Operators

A page description begins with an empty path and builds up its definition by invoking one or more path construction operators to add segments to it. The path construction operators may be invoked in any sequence, but the first one invoked must be **m** or **re** to begin a new subpath. The path definition concludes with the application of a path-painting operator such as **S**, **f**, or **b** (see Section 4.4.2, "Path-Painting Operators"); this operator may optionally be preceded by one of the clipping path operators **W** or **W*** (Section 4.4.3, "Clipping Path Operators"). Note that the path construction operators do not place any marks on the page; only the painting operators do that. A path definition is not complete until a path-painting operator has been applied to it.

The path currently under construction is called the *current path*. In PDF (unlike PostScript), the current path is *not* part of the graphics state and is *not* saved and restored along with the other graphics state parameters. PDF paths are strictly internal objects with no explicit representation. Once a path has been painted, it is no longer defined; there is then no current path until a new one is begun with the **m** or **re** operator.

The trailing endpoint of the segment most recently added to the current path is referred to as the *current point*. If the current path is empty, the current point is undefined. Most operators that add a segment to the current path start at the current point; if the current point is undefined, an error is generated.

Table 4.9 shows the path construction operators. All operands are numbers denoting coordinates in user space.

TABLE 4.9 Path construction operators

OPERANDS	OPERATOR	DESCRIPTION
$x\ y$	**m**	Begin a new subpath by moving the current point to coordinates (x, y), omitting any connecting line segment. If the previous path construction operator in the current path was also **m**, the new **m** overrides it; no vestige of the previous **m** operation remains in the path.
$x\ y$	**l** (lowercase **L**)	Append a straight line segment from the current point to the point (x, y). The new current point is (x, y).
$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$	**c**	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3), using (x_1, y_1) and (x_2, y_2) as the Bézier control points (see "Cubic Bézier Curves," below). The new current point is (x_3, y_3).
$x_2\ y_2\ x_3\ y_3$	**v**	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3), using the current point and (x_2, y_2) as the Bézier control points (see "Cubic Bézier Curves," below). The new current point is (x_3, y_3).
$x_1\ y_1\ x_3\ y_3$	**y**	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3), using (x_1, y_1) and (x_3, y_3) as the Bézier control points (see "Cubic Bézier Curves," below). The new current point is (x_3, y_3).

OPERANDS	OPERATOR	DESCRIPTION
—	h	Close the current subpath by appending a straight line segment from the current point to the starting point of the subpath. If the current subpath is already closed, h does nothing. This operator terminates the current subpath. Appending another segment to the current path begins a new subpath, even if the new segment begins at the endpoint reached by the h operation.
x y width height	re	Append a rectangle to the current path as a complete subpath, with lower-left corner (x, y) and dimensions *width* and *height* in user space. The operation *x y width height* re is equivalent to *x y* m *(x + width) y* l *(x + width) (y + height)* l *x (y + height)* l h

Cubic Bézier Curves

Curved path segments are specified as *cubic Bézier curves*. Such curves are defined by four points: the two endpoints (the current point P_0 and the final point P_3) and two *control points* P_1 and P_2. Given the coordinates of the four points, the curve is generated by varying the parameter t from 0.0 to 1.0 in the following equation:

$$R(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

When $t = 0.0$, the value of the function $R(t)$ coincides with the current point P_0; when $t = 1.0$, $R(t)$ coincides with the final point P_3. Intermediate values of t generate intermediate points along the curve. The curve does not, in general, pass through the two control points P_1 and P_2.

Cubic Bézier curves have two useful properties:

- The curve can be very quickly split into smaller pieces for rapid rendering.
- The curve is contained within the convex hull of the four points defining the curve, most easily visualized as the polygon obtained by stretching a rubber band around the outside of the four points. This property allows rapid testing of whether the curve lies completely outside the visible region, and hence does not have to be rendered.

The Bibliography lists several books that describe cubic Bézier curves in more depth.

The most general PDF operator for constructing curved path segments is the **c** operator, which specifies the coordinates of points P_1, P_2, and P_3 explicitly, as shown in Figure 4.8. (The starting point, P_0, is defined implicitly by the current point.)

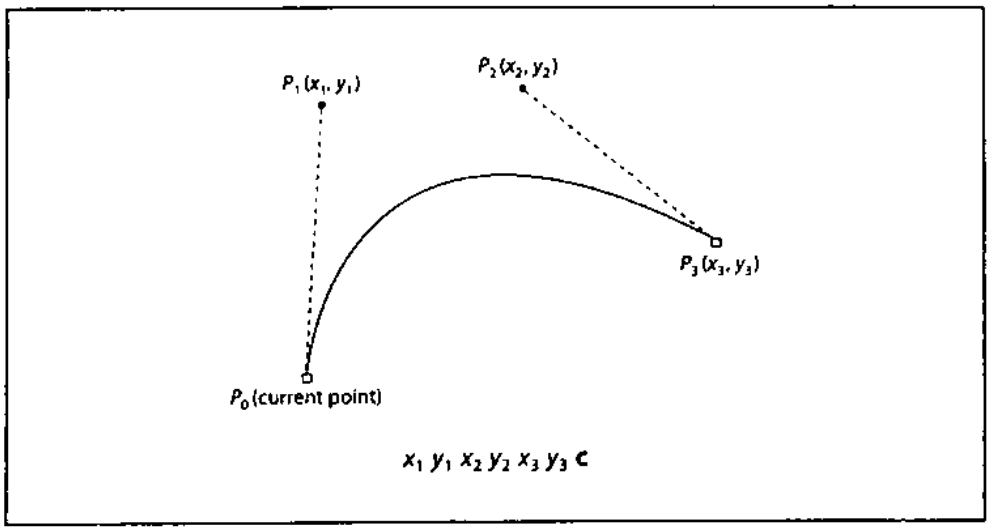


FIGURE 4.8 *Cubic Bézier curve generated by the* **c** *operator*

Two more operators, **v** and **y**, each specify one of the two control points implicitly (see Figure 4.9). In both of these cases, one control point and the final point of the curve are supplied as operands; the other control point is implied:

- For the **v** operator, the first control point coincides with initial point of the curve.
- For the **y** operator, the second control point coincides with final point of the curve.

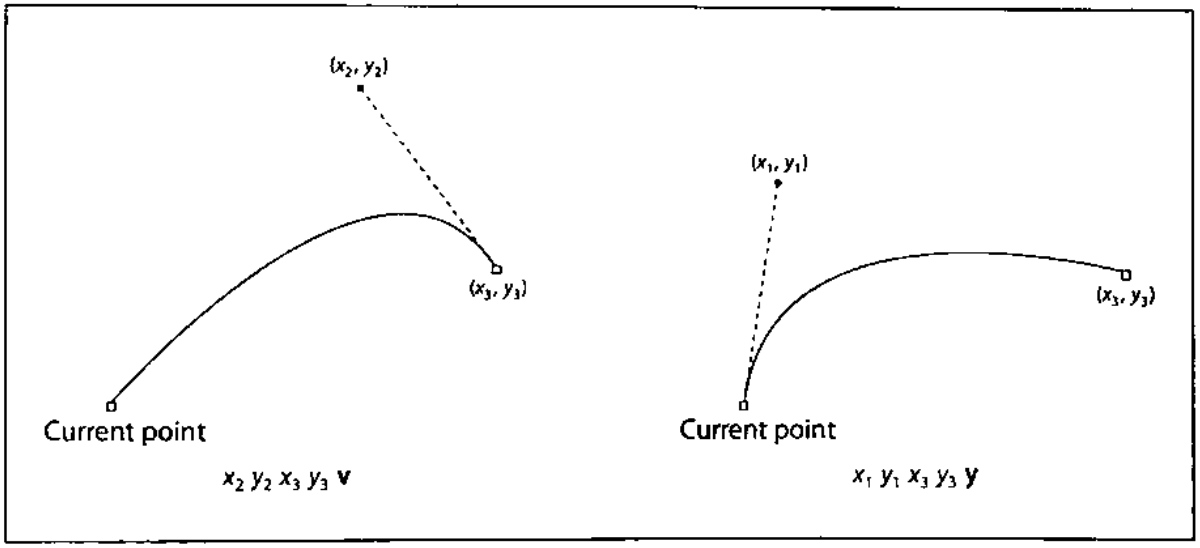


FIGURE 4.9 *Cubic Bézier curves generated by the* **v** *and* **y** *operators*

4.4.2 Path-Painting Operators

The path-painting operators end a path object, causing it to be painted on the current page in the manner that the operator specifies. The principal path-painting operators are **S** (for *stroking*) and **f** (for *filling*). Variants of these operators combine stroking and filling in a single operation or apply different rules for determining the area to be filled. Table 4.10 lists all the path-painting operators.

TABLE 4.10 Path-painting operators

OPERANDS	OPERATOR	DESCRIPTION
—	S	Stroke the path.
—	s	Close and stroke the path. This operator has the same effect as the sequence h S.
—	f	Fill the path, using the nonzero winding number rule to determine the region to fill (see "Nonzero Winding Number Rule" on page 232). Any subpaths that are open are implicitly closed before being filled.
—	F	Equivalent to f; included only for compatibility. Although PDF consumer applications must be able to accept this operator, PDF producer applications should use f instead.
—	f*	Fill the path, using the even-odd rule to determine the region to fill (see "Even-Odd Rule" on page 233).
—	B	Fill and then stroke the path, using the nonzero winding number rule to determine the region to fill. This operator produces the same result as constructing two identical path objects, painting the first with f and the second with S. Note, however, that the filling and stroking portions of the operation consult different values of several graphics state parameters, such as the current color. See also "Special Path-Painting Considerations" on page 569.
—	B*	Fill and then stroke the path, using the even-odd rule to determine the region to fill. This operator produces the same result as B, except that the path is filled as if with f* instead of f. See also "Special Path-Painting Considerations" on page 569.
—	b	Close, fill, and then stroke the path, using the nonzero winding number rule to determine the region to fill. This operator has the same effect as the sequence h B. See also "Special Path-Painting Considerations" on page 569.
—	b*	Close, fill, and then stroke the path, using the even-odd rule to determine the region to fill. This operator has the same effect as the sequence h B*. See also "Special Path-Painting Considerations" on page 569.
—	n	End the path object without filling or stroking it. This operator is a path-painting no-op, used primarily for the side effect of changing the current clipping path (see Section 4.4.3, "Clipping Path Operators").

Stroking

The **S** operator paints a line along the current path. The stroked line follows each straight or curved segment in the path, centered on the segment with sides parallel to it. Each of the path's subpaths is treated separately.

The results of the **S** operator depend on the current settings of various parameters in the graphics state (see Section 4.3, "Graphics State," for further information on these parameters):

- The width of the stroked line is determined by the current line width parameter ("Line Width" on page 215).
- The color or pattern of the line is determined by the current color and color space for stroking operations.
- The line can be painted either solid or with a dash pattern, as specified by the current line dash pattern ("Line Dash Pattern" on page 217).
- If a subpath is open, the unconnected ends are treated according to the current line cap style, which may be butt, rounded, or square ("Line Cap Style" on page 216).
- Wherever two consecutive segments are connected, the joint between them is treated according to the current line join style, which may be mitered, rounded, or beveled ("Line Join Style" on page 216). Mitered joins are also subject to the current miter limit ("Miter Limit" on page 217).

  Note: Points at which unconnected segments happen to meet or intersect receive no special treatment. In particular, using an explicit **l** *operator to give the appearance of closing a subpath, rather than using* **h**, *may result in a messy corner, because line caps are applied instead of a line join.*

- The stroke adjustment parameter *(PDF 1.2)* specifies that coordinates and line widths be adjusted automatically to produce strokes of uniform thickness despite rasterization effects (Section 6.5.4, "Automatic Stroke Adjustment").

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), the **S** operator paints it only if round line caps have been specified, producing a filled circle centered at the single point. If butt or projecting square line caps have been specified, **S** produces no output, because the orientation of the caps would be indeterminate. (This rule applies only to zero-length subpaths of the path being stroked, and not to zero-length dashes

in a dash pattern. In the latter case, the line caps are always painted, since their orientation is determined by the direction of the underlying path.) A single-point open subpath (specified by a trailing **m** operator) produces no output.

Filling

The **f** operator uses the current nonstroking color to paint the entire region enclosed by the current path. If the path consists of several disconnected subpaths, **f** paints the insides of all subpaths, considered together. Any subpaths that are open are implicitly closed before being filled.

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), **f** paints the single device pixel lying under that point; the result is device-dependent and not generally useful. A single-point open subpath (specified by a trailing **m** operator) produces no output.

For a simple path, it is intuitively clear what region lies inside. However, for a more complex path—for example, a path that intersects itself or has one subpath that encloses another—it is not always obvious which points lie inside the path. The path machinery uses one of two rules for determining which points lie inside a path: the *nonzero winding number rule* and the *even-odd rule*, both discussed in detail below.

The nonzero winding number rule is more versatile than the even-odd rule and is the standard rule the **f** operator uses. Similarly, the **W** operator uses this rule to determine the inside of the current clipping path. The even-odd rule is occasionally useful for special effects or for compatibility with other graphics systems; the **f*** and **W*** operators invoke this rule.

Nonzero Winding Number Rule

The *nonzero winding number rule* determines whether a given point is inside a path by conceptually drawing a ray from that point to infinity in any direction and then examining the places where a segment of the path crosses the ray. Starting with a count of 0, the rule adds 1 each time a path segment crosses the ray from left to right and subtracts 1 each time a segment crosses from right to left. After counting all the crossings, if the result is 0, the point is outside the path; otherwise, it is inside.

Note: The method just described does not specify what to do if a path segment coincides with or is tangent to the chosen ray. Since the direction of the ray is arbitrary, the rule simply chooses a ray that does not encounter such problem intersections.

For simple convex paths, the nonzero winding number rule defines the inside and outside as one would intuitively expect. The more interesting cases are those involving complex or self-intersecting paths like the ones shown in Figure 4.10. For a path consisting of a five-pointed star, drawn with five connected straight line segments intersecting each other, the rule considers the inside to be the entire area enclosed by the star, including the pentagon in the center. For a path composed of two concentric circles, the areas enclosed by both circles are considered to be inside, *provided that both are drawn in the same direction.* If the circles are drawn in opposite directions, only the doughnut shape between them is inside, according to the rule; the doughnut hole is outside.

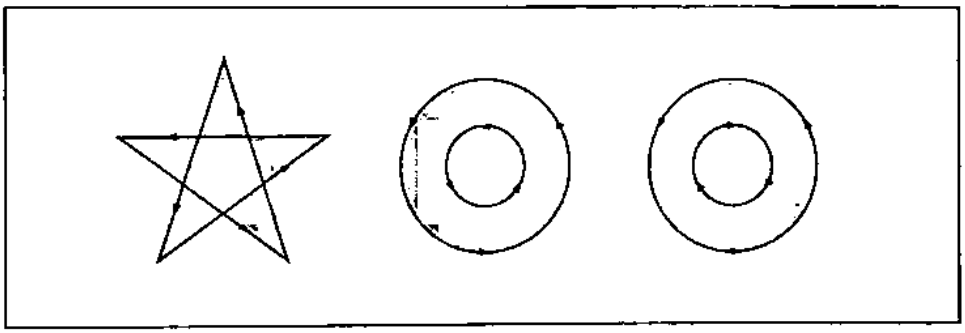


FIGURE 4.10 *Nonzero winding number rule*

Even-Odd Rule

An alternative to the nonzero winding number rule is the *even-odd rule.* This rule determines whether a point is inside a path by drawing a ray from that point in any direction and simply counting the number of path segments that cross the ray, regardless of direction. If this number is odd, the point is inside; if even, the point is outside. This yields the same results as the nonzero winding number rule for paths with simple shapes, but produces different results for more complex shapes.

Figure 4.11 shows the effects of applying the even-odd rule to complex paths. For the five-pointed star, the rule considers the triangular points to be inside the path,

but not the pentagon in the center. For the two concentric circles, only the doughnut shape between the two circles is considered inside, regardless of the directions in which the circles are drawn.

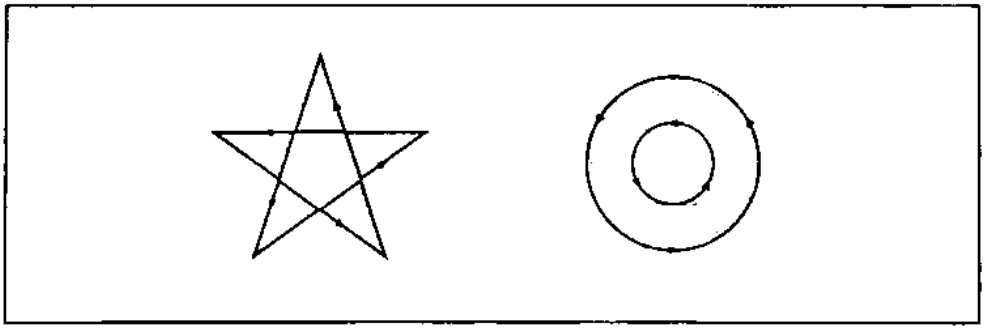


FIGURE 4.11 *Even-odd rule*

4.4.3 Clipping Path Operators

The graphics state contains a *current clipping path* that limits the regions of the page affected by painting operators. The closed subpaths of this path define the area that can be painted. Marks falling inside this area are applied to the page; those falling outside it are not. ("Filling" on page 232 discusses precisely what is considered to be inside a path.)

Note: In the context of the transparent imaging model (PDF 1.4), *the current clipping path constrains an object's shape (see Section 7.1, "Overview of Transparency"). The effective shape is the intersection of the object's intrinsic shape with the clipping path; the source shape value is 0.0 outside this intersection. Similarly, the shape of a transparency group (defined as the union of the shapes of its constituent objects) is influenced both by the clipping path in effect when each of the objects is painted and by the one in effect at the time the group's results are painted onto its backdrop.*

The initial clipping path includes the entire page. A clipping path operator (**W** or **W***, shown in Table 4.11) may appear after the last path construction operator and before the path-painting operator that terminates a path object. Although the clipping path operator appears before the painting operator, it does not alter the clipping path at the point where it appears. Rather, it modifies the effect of the succeeding painting operator. *After* the path has been painted, the clipping path in the graphics state is set to the intersection of the current clipping path and the newly constructed path.

TABLE 4.11 Clipping path operators

OPERANDS	OPERATOR	DESCRIPTION
—	W	Modify the current clipping path by intersecting it with the current path, using the nonzero winding number rule to determine which regions lie inside the clipping path.
—	W*	Modify the current clipping path by intersecting it with the current path, using the even-odd rule to determine which regions lie inside the clipping path.

Note: In addition to path objects, text objects can also be used for clipping; see Section 5.2.5, "Text Rendering Mode."

The n operator (see Table 4.10) is a no-op path-painting operator; it causes no marks to be placed on the page, but can be used with a clipping path operator to establish a new clipping path. That is, after a path has been constructed, the sequence W n intersects that path with the current clipping path to establish a new clipping path.

There is no way to enlarge the current clipping path or to set a new clipping path without reference to the current one. However, since the clipping path is part of the graphics state, its effect can be localized to specific graphics objects by enclosing the modification of the clipping path and the painting of those objects between a pair of q and Q operators (see Section 4.3.1, "Graphics State Stack"). Execution of the Q operator causes the clipping path to revert to the value that was saved by the q operator before the clipping path was modified.

4.5 Color Spaces

PDF includes powerful facilities for specifying the colors of graphics objects to be painted on the current page. The color facilities are divided into two parts:

- *Color specification.* A PDF file can specify abstract colors in a device-independent way. Colors can be described in any of a variety of color systems, or *color spaces*. Some color spaces are related to device color representation (grayscale, *RGB*, *CMYK*), others to human visual perception (CIE-based). Certain special features are also modeled as color spaces: patterns, color mapping, separations, and high-fidelity and multitone color.

- *Color rendering.* The application reproduces colors on the raster output device by a multiple-step process that includes some combination of color conversion, gamma correction, halftoning, and scan conversion. Some aspects of this process use information that is specified in PDF. However, unlike the facilities for color specification, the color-rendering facilities are device-dependent and ordinarily should not be included in a page description.

Figures 4.12 and 4.13 on pages 238 and 239 illustrate the division between PDF's (device-independent) color specification and (device-dependent) color-rendering facilities. This section describes the color specification features, covering everything that most PDF documents need to specify colors. The facilities for controlling color rendering are described in Chapter 6; a PDF document should use these facilities only to configure or calibrate an output device or to achieve special device-dependent effects.

4.5.1 Color Values

As described in Section 4.4.2, "Path-Painting Operators," marks placed on the page by operators such as **f** and **S** have a color that is determined by the *current color* parameter of the graphics state. A color value consists of one or more *color components*, which are usually numbers. For example, a gray level can be specified by a single number ranging from 0.0 (black) to 1.0 (white). Full color values can be specified in any of several ways; a common method uses three numeric values to specify red, green, and blue components.

Color values are interpreted according to the *current color space*, another parameter of the graphics state. A PDF content stream first selects a color space by invoking the **CS** operator (for the stroking color) or the **cs** operator (for the nonstroking color). It then selects color values within that color space with the **SC** operator (stroking) or the **sc** operator (nonstroking). There are also convenience operators—**G**, **g**, **RG**, **rg**, **K**, and **k**—that select both a color space and a color value within it in a single step. Table 4.24 on page 287 lists all the color-setting operators.

Sampled images (see Section 4.8, "Images") specify the color values of individual samples with respect to a color space designated by the image object itself. While these values are independent of the current color space and color parameters in the graphics state, all later stages of color processing treat them in exactly the same way as color values specified with the **SC** or **sc** operator.

4.5.2 Color Space Families

Color spaces can be classified into *color space families*. Spaces within a family share the same general characteristics; they are distinguished by parameter values supplied at the time the space is specified. The families fall into three broad categories:

- *Device color spaces* directly specify colors or shades of gray that the output device is to produce. They provide a variety of color specification methods, including grayscale, *RGB* (red-green-blue), and *CMYK* (cyan-magenta-yellow-black), corresponding to the color space families **DeviceGray**, **DeviceRGB**, and **DeviceCMYK**. Since each of these families consists of just a single color space with no parameters, they are often loosely referred to as the **DeviceGray**, **DeviceRGB**, and **DeviceCMYK** color spaces.
- *CIE-based color spaces* are based on an international standard for color specification created by the Commission Internationale de l'Éclairage (International Commission on Illumination). These spaces specify colors in a way that is independent of the characteristics of any particular output device. Color space families in this category include **CalGray**, **CalRGB**, **Lab**, and **ICCBased**. Individual color spaces within these families are specified by means of dictionaries containing the parameter values needed to define the space.
- *Special color spaces* add features or properties to an underlying color space. They include facilities for patterns, color mapping, separations, and high-fidelity and multitone color. The corresponding color space families are **Pattern**, **Indexed**, **Separation**, and **DeviceN**. Individual color spaces within these families are specified by means of additional parameters.

Table 4.12 summarizes the color space families supported by PDF. (See implementation note 47 in Appendix H.)

TABLE 4.12 Color space families

DEVICE	CIE-BASED	SPECIAL
DeviceGray *(PDF 1.1)*	**CalGray** *(PDF 1.1)*	**Indexed** *(PDF 1.1)*
DeviceRGB *(PDF 1.1)*	**CalRGB** *(PDF 1.1)*	**Pattern** *(PDF 1.2)*
DeviceCMYK *(PDF 1.1)*	**Lab** *(PDF 1.1)*	**Separation** *(PDF 1.2)*
	ICCBased *(PDF 1.3)*	**DeviceN** *(PDF 1.3)*

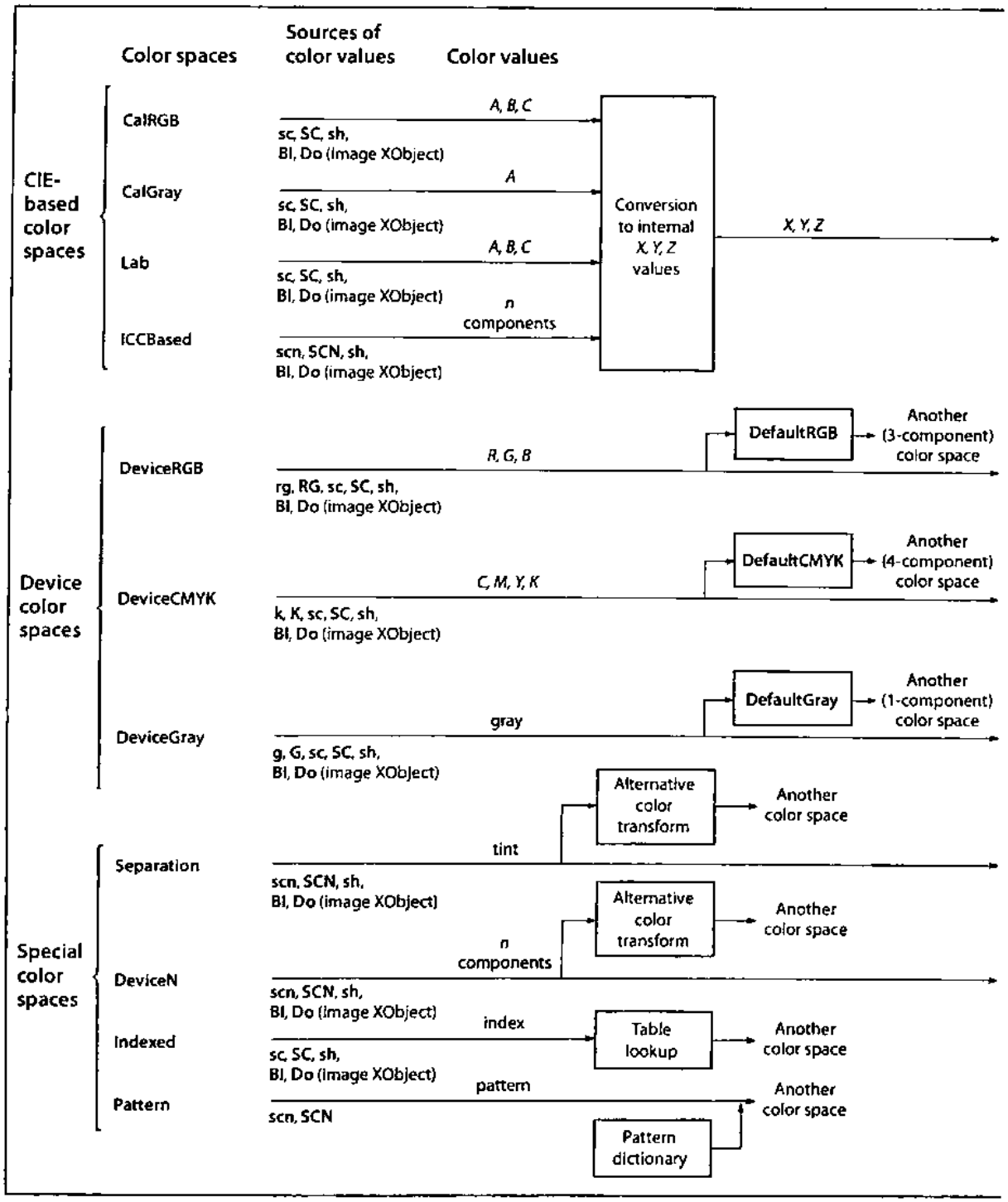


FIGURE 4.12 *Color specification*

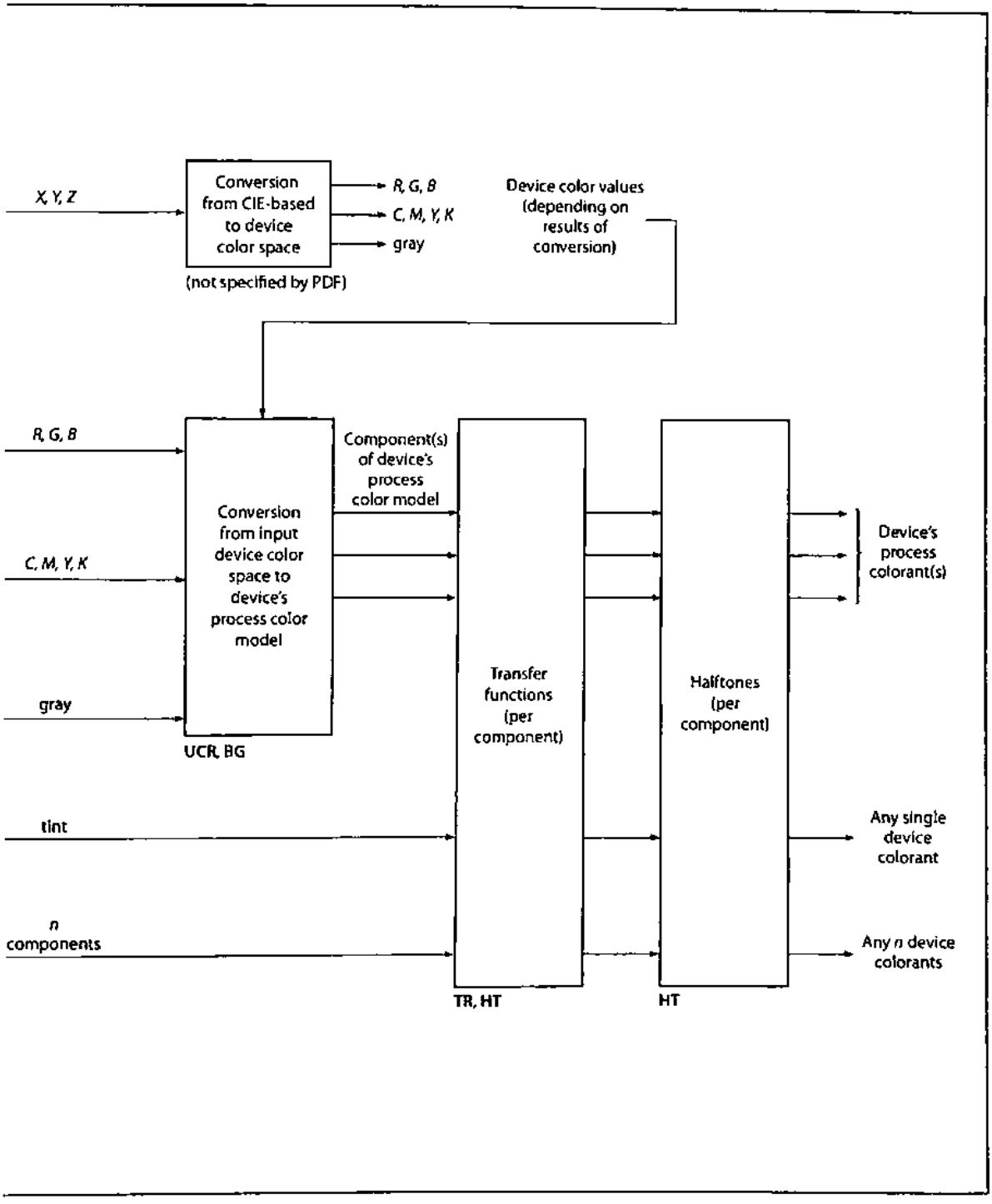


FIGURE 4.13 *Color rendering*

A color space is defined by an array object whose first element is a name object identifying the color space family. The remaining array elements, if any, are parameters that further characterize the color space; their number and types vary according to the particular family. For families that do not require parameters, the color space can be specified simply by the family name itself instead of an array.

A color space can be specified in two principal ways:

- Within a content stream, the **CS** or **cs** operator establishes the current color space parameter in the graphics state. The operand is always a name object, which either identifies one of the color spaces that need no additional parameters (**DeviceGray**, **DeviceRGB**, **DeviceCMYK**, or some cases of **Pattern**) or is used as a key in the **ColorSpace** subdictionary of the current resource dictionary (see Section 3.7.2, "Resource Dictionaries"). In the latter case, the value of the dictionary entry is in turn a color space array or name. A color space array is never permitted inline within a content stream.
- Outside a content stream, certain objects, such as image XObjects, specify a color space as an explicit parameter, often associated with the key **ColorSpace**. In this case, the color space array or name is always defined directly as a PDF object, not by an entry in the **ColorSpace** resource subdictionary. This convention also applies when color spaces are defined in terms of other color spaces.

The following operators set the current color space and current color parameters in the graphics state:

- **CS** sets the stroking color space; **cs** sets the nonstroking color space.
- **SC** and **SCN** set the stroking color; **sc** and **scn** set the nonstroking color. Depending on the color space, these operators require one or more operands, each specifying one component of the color value.
- **G**, **RG**, and **K** set the stroking color space implicitly and the stroking color as specified by the operands; **g**, **rg**, and **k** do the same for the nonstroking color space and color.

4.5.3 Device Color Spaces

The device color spaces enable a page description to specify color values that are directly related to their representation on an output device. Color values in these spaces map directly (or by simple conversions) to the application of device colorants, such as quantities of ink or intensities of display phosphors. This enables a PDF document to control colors precisely for a particular device, but the results may not be consistent from one device to another.

Output devices form colors either by adding light sources together or by subtracting light from an illuminating source. Computer displays and film recorders typically add colors, printing inks typically subtract them. These two ways of forming colors give rise to two complementary methods of color specification, called *additive* and *subtractive* color (see Plate 1). The most widely used forms of these two types of color specification are known as *RGB* and *CMYK*, respectively, for the names of the primary colors on which they are based. They correspond to the following device color spaces:

- **DeviceGray** controls the intensity of achromatic light, on a scale from black to white.
- **DeviceRGB** controls the intensities of red, green, and blue light, the three additive primary colors used in displays.
- **DeviceCMYK** controls the concentrations of cyan, magenta, yellow, and black inks, the four subtractive process colors used in printing.

Although the notion of explicit color spaces is a PDF 1.1 feature, the operators for specifying colors in the device color spaces—**G**, **g**, **RG**, **rg**, **K**, and **k**—are available in all versions of PDF. Beginning with PDF 1.2, colors specified in device color spaces can optionally be remapped systematically into other color spaces; see "Default Color Spaces" on page 257.

Note: In the transparent imaging model (PDF 1.4), *the use of device color spaces is subject to special treatment within a transparency group whose group color space is CIE-based (see Sections 7.3, "Transparency Groups," and 7.5.5, "Transparency Group XObjects"). In particular, the device color space operators should be used only if device color spaces have been remapped to CIE-based spaces by means of the default color space mechanism. Otherwise, the results are implementation-dependent and unpredictable.*

DeviceGray Color Space

Black, white, and intermediate shades of gray are special cases of full color. A grayscale value is represented by a single number in the range 0.0 to 1.0, where 0.0 corresponds to black, 1.0 to white, and intermediate values to different gray levels. Example 4.2 shows alternative ways to select the **DeviceGray** color space and a specific gray level within that space for stroking operations.

Example 4.2

```
/DeviceGray CS          % Set DeviceGray color space
gray SC                 % Set gray level

gray G                  % Set both in one operation
```

The **CS** and **SC** operators select the current stroking color space and current stroking color separately; **G** sets them in combination. (The **cs**, **sc**, and **g** operators perform the same functions for nonstroking operations.) Setting either current color space to **DeviceGray** initializes the corresponding current color to 0.0.

DeviceRGB Color Space

Colors in the **DeviceRGB** color space are specified according to the additive *RGB* (red-green-blue) color model, in which color values are defined by three components representing the intensities of the additive primary colorants red, green, and blue. Each component is specified by a number in the range 0.0 to 1.0, where 0.0 denotes the complete absence of a primary component and 1.0 denotes maximum intensity. If all three components have equal intensity, the perceived result theoretically is a pure gray on the scale from black to white. If the intensities are not all equal, the result is some color other than a pure gray.

Example 4.3 shows alternative ways to select the **DeviceRGB** color space and a specific color within that space for stroking operations.

Example 4.3

```
/DeviceRGB CS           % Set DeviceRGB color space
red green blue SC       % Set color

red green blue RG       % Set both in one operation
```

The **CS** and **SC** operators select the current stroking color space and current stroking color separately; **RG** sets them in combination. (The **cs**, **sc**, and **rg** operators perform the same functions for nonstroking operations.) Setting either current color space to **DeviceRGB** initializes the red, green, and blue components of the corresponding current color to 0.0.

DeviceCMYK Color Space

The **DeviceCMYK** color space allows colors to be specified according to the subtractive *CMYK* (cyan-magenta-yellow-black) model typical of printers and other paper-based output devices. In theory, each of the three standard *process colorants* used in printing (cyan, magenta, and yellow) absorbs one of the additive primary colors (red, green, and blue, respectively). Black, a fourth standard process colorant, absorbs all of the additive primaries in equal amounts. The four components in a **DeviceCMYK** color value represent the concentrations of these process colorants. Each component is specified by a number in the range 0.0 to 1.0, where 0.0 denotes the complete absence of a process colorant (that is, absorbs none of the corresponding additive primary) and 1.0 denotes maximum concentration (absorbs as much as possible of the additive primary). Note that the sense of these numbers is opposite to that of *RGB* color components.

Example 4.4 shows alternative ways to select the **DeviceCMYK** color space and a specific color within that space for stroking operations.

Example 4.4

```
/DeviceCMYK CS                    % Set DeviceCMYK color space
cyan magenta yellow black SC      % Set color

cyan magenta yellow black K       % Set both in one operation
```

The **CS** and **SC** operators select the current stroking color space and current stroking color separately; **K** sets them in combination. (The **cs**, **sc**, and **k** operators perform the same functions for nonstroking operations.) Setting either current color space to **DeviceCMYK** initializes the cyan, magenta, and yellow components of the corresponding current color to 0.0 and the black component to 1.0.

4.5.4 CIE-Based Color Spaces

Calibrated color in PDF is defined in terms of an international standard used in the graphic arts, television, and printing industries. *CIE-based* color spaces enable a page description to specify color values in a way that is related to human visual perception. The goal is for the same color specification to produce consistent results on different output devices, within the limitations of each device; Plate 2 illustrates the kind of variation in color reproduction that can result from the use of uncalibrated color on different devices. PDF 1.1 supports three CIE-based color space families, named **CalGray**, **CalRGB**, and **Lab**; PDF 1.3 adds a fourth, named **ICCBased**.

Note: In PDF 1.1, a color space family named ***CalCMYK*** *was partially defined, with the expectation that its definition would be completed in a future version. However, this is no longer being considered. PDF 1.3 and later versions support calibrated four-component color spaces by means of ICC profiles (see "ICCBased Color Spaces" on page 252). PDF consumer applications should ignore* ***CalCMYK*** *color space attributes and render colors specified in this family as if they had been specified using* ***DeviceCMYK****.*

The details of the CIE colorimetric system and the theory on which it is based are beyond the scope of this book; see the Bibliography for sources of further information. The semantics of CIE-based color spaces are defined in terms of the relationship between the space's components and the tristimulus values *X*, *Y*, and *Z* of the CIE 1931 *XYZ* space. The **CalRGB** and **Lab** color spaces *(PDF 1.1)* are special cases of three-component CIE-based color spaces, known as *CIE-based ABC* color spaces. These spaces are defined in terms of a two-stage, nonlinear transformation of the CIE 1931 *XYZ* space. The formulation of such color spaces models a simple *zone theory* of color vision, consisting of a nonlinear trichromatic first stage combined with a nonlinear opponent-color second stage. This formulation allows colors to be digitized with minimum loss of fidelity, an important consideration in sampled images.

Color values in a CIE-based *ABC* color space have three components, arbitrarily named *A*, *B*, and *C*. The first stage transforms these components by first forcing their values to a specified range, then applying *decoding functions*, and then multiplying the results by a 3-by-3 matrix, producing three intermediate components arbitrarily named *L*, *M*, and *N*. The second stage transforms these intermediate components in a similar fashion, producing the final *X*, *Y*, and *Z* components of the CIE 1931 *XYZ* space (see Figure 4.14).

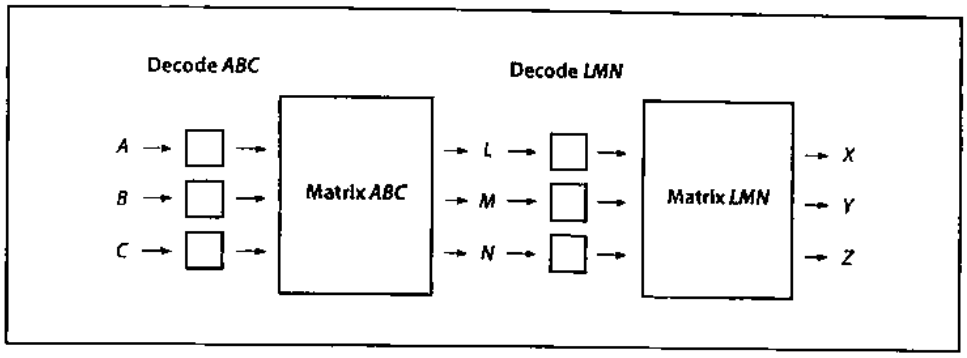


FIGURE 4.14 *Component transformations in a CIE-based* ABC *color space*

Color spaces in the CIE-based families are defined by an array

[*name dictionary*]

where *name* is the name of the family and *dictionary* is a dictionary containing parameters that further characterize the space. The entries in this dictionary have specific interpretations that depend on the color space; some entries are required and some are optional. See the sections on specific color space families, below, for details.

Setting the current stroking or nonstroking color space to any CIE-based color space initializes all components of the corresponding current color to 0.0 (unless the range of valid values for a given component does not include 0.0, in which case the nearest valid value is substituted.)

Note: *The model and terminology used here—*CIE-based ABC *(above) and* CIE-based A *(below)—are derived from the PostScript language, which supports these color space families in their full generality. PDF supports specific useful cases of* CIE-based ABC *and* CIE-based A *spaces; most others can be represented as* **ICCBased** *spaces.*

CalGray Color Spaces

A **CalGray** color space *(PDF 1.1)* is a special case of a single-component CIE-based color space, known as a *CIE-based A* color space. This type of space is the one-dimensional (and usually achromatic) analog of CIE-based *ABC* spaces. Color values in a CIE-based *A* space have a single component, arbitrarily named *A*. Figure 4.15 illustrates the transformations of the *A* component to *X*, *Y*, and *Z* components of the CIE 1931 *XYZ* space.

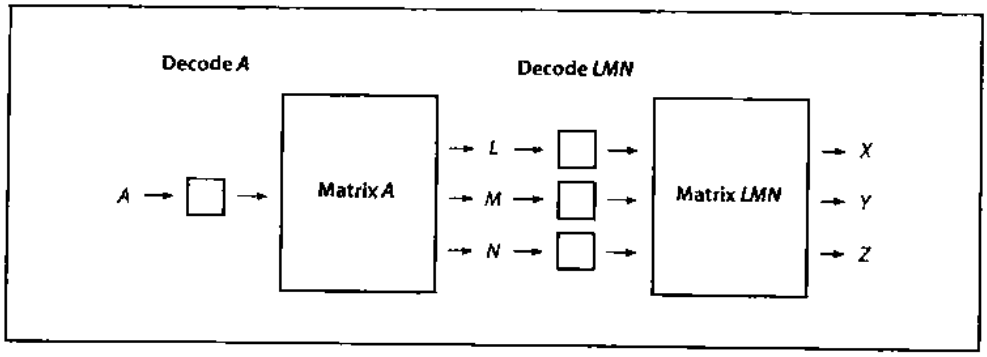


FIGURE 4.15 *Component transformations in a CIE-based* A *color space*

A **CalGray** color space is a CIE-based *A* color space with only one transformation stage instead of two. In this type of space, *A* represents the gray component of a calibrated gray space. This component must be in the range 0.0 to 1.0. The decoding function (denoted by "Decode *A*" in Figure 4.15) is a gamma function whose coefficient is specified by the **Gamma** entry in the color space dictionary (see Table 4.13). The transformation matrix denoted by "Matrix *A*" in the figure is derived from the dictionary's **WhitePoint** entry, as described below. Since there is no second transformation stage, "Decode *LMN*" and "Matrix *LMN*" are implicitly taken to be identity transformations.

TABLE 4.13 Entries in a CalGray color space dictionary

KEY	TYPE	VALUE
WhitePoint	array	*(Required)* An array of three numbers $[X_W\ Y_W\ Z_W]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see "CalRGB Color Spaces," below, for further discussion. The numbers X_W and Z_W must be positive, and Y_W must be equal to 1.0.
BlackPoint	array	*(Optional)* An array of three numbers $[X_B\ Y_B\ Z_B]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see "CalRGB Color Spaces," below, for further discussion. All three of these numbers must be non-negative. Default value: [0.0 0.0 0.0].
Gamma	number	*(Optional)* A number *G* defining the gamma for the gray *(A)* component. *G* must be positive and is generally greater than or equal to 1. Default value: 1.

The transformation defined by the **Gamma** and **WhitePoint** entries is

$$X = L = X_W \times A^G$$
$$Y = M = Y_W \times A^G$$
$$Z = N = Z_W \times A^G$$

In other words, the A component is first decoded by the gamma function, and the result is multiplied by the components of the white point to obtain the L, M, and N components of the intermediate representation. Since there is no second stage, the L, M, and N components are also the X, Y, and Z components of the final representation.

The following examples illustrate interesting and useful special cases of **CalGray** spaces. Example 4.5 establishes a space consisting of the Y dimension of the CIE 1931 XYZ space with the CCIR XA/11–recommended D65 white point.

Example 4.5

```
[ /CalGray
      << /WhitePoint [0.9505 1.0000 1.0890] >>
]
```

Example 4.6 establishes a calibrated gray space with the CCIR XA/11–recommended D65 white point and opto-electronic transfer function.

Example 4.6

```
[ /CalGray
      << /WhitePoint [0.9505 1.0000 1.0890]
         /Gamma 2.222
      >>
]
```

CalRGB Color Spaces

A **CalRGB** color space is a CIE-based ABC color space with only one transformation stage instead of two. In this type of space, A, B, and C represent calibrated red, green, and blue color values. These three color components must be in the range 0.0 to 1.0; component values falling outside that range are adjusted to the nearest valid value without error indication. The decoding functions (denoted by "Decode ABC" in Figure 4.14 on page 245) are gamma functions whose coeffi-

cients are specified by the **Gamma** entry in the color space dictionary (see Table 4.14). The transformation matrix denoted by "Matrix *ABC*" in Figure 4.14 is defined by the dictionary's **Matrix** entry. Since there is no second transformation stage, "Decode *LMN*" and "Matrix *LMN*" are implicitly taken to be identity transformations.

TABLE 4.14 Entries in a CalRGB color space dictionary

KEY	TYPE	VALUE
WhitePoint	array	*(Required)* An array of three numbers $[X_W\ Y_W\ Z_W]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see below for further discussion. The numbers X_W and Z_W must be positive, and Y_W must be equal to 1.0.
BlackPoint	array	*(Optional)* An array of three numbers $[X_B\ Y_B\ Z_B]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see below for further discussion. All three of these numbers must be non-negative. Default value: [0.0 0.0 0.0].
Gamma	array	*(Optional)* An array of three numbers $[G_R\ G_G\ G_B]$ specifying the gamma for the red, green, and blue *(A, B,* and *C)* components of the color space. Default value: [1.0 1.0 1.0].
Matrix	array	*(Optional)* An array of nine numbers $[X_A\ Y_A\ Z_A\ X_B\ Y_B\ Z_B\ X_C\ Y_C\ Z_C]$ specifying the linear interpretation of the decoded *A, B,* and *C* components of the color space with respect to the final *XYZ* representation. Default value: the identity matrix [1 0 0 0 1 0 0 0 1].

The **WhitePoint** and **BlackPoint** entries in the color space dictionary control the overall effect of the CIE-based gamut mapping function described in Section 6.1, "CIE-Based Color to Device Color." Typically, the colors specified by **WhitePoint** and **BlackPoint** are mapped to the nearly lightest and nearly darkest achromatic colors that the output device is capable of rendering in a way that preserves color appearance and visual contrast.

WhitePoint is assumed to represent the diffuse achromatic highlight, not a specular highlight. Specular highlights, achromatic or otherwise, are often reproduced lighter than the diffuse highlight. **BlackPoint** is assumed to represent the diffuse achromatic shadow; its value is typically limited by the dynamic range of the input device. In images produced by a photographic system, the values of **WhitePoint** and **BlackPoint** vary with exposure, system response, and artistic intent; hence, their values are image-dependent.

The transformation defined by the **Gamma** and **Matrix** entries in the **CalRGB** color space dictionary is

$$X = L = X_A \times A^{G_R} + X_B \times B^{G_G} + X_C \times C^{G_B}$$
$$Y = M = Y_A \times A^{G_R} + Y_B \times B^{G_G} + Y_C \times C^{G_B}$$
$$Z = N = Z_A \times A^{G_R} + Z_B \times B^{G_G} + Z_C \times C^{G_B}$$

In other words, the *A*, *B*, and *C* components are first decoded individually by the gamma functions. The results are treated as a three-element vector and multiplied by **Matrix** (a 3-by-3 matrix) to obtain the *L*, *M*, and *N* components of the intermediate representation. Since there is no second stage, these are also the *X*, *Y*, and *Z* components of the final representation.

Example 4.7 shows an example of a **CalRGB** color space for the CCIR XA/11–recommended D65 white point with 1.8 gammas and Sony Trinitron phosphor chromaticities.

Example 4.7

```
[ /CalRGB
      << /WhitePoint [0.9505 1.0000 1.0890]
         /Gamma [1.8000 1.8000 1.8000]
         /Matrix [ 0.4497 0.2446 0.0252
                   0.3163 0.6720 0.1412
                   0.1845 0.0833 0.9227
                 ]
      >>
]
```

In some cases, the parameters of a **CalRGB** color space may be specified in terms of the CIE 1931 chromaticity coordinates (x_R, y_R), (x_G, y_G), (x_B, y_B) of the red, green, and blue phosphors, respectively, and the chromaticity (x_W, y_W) of the diffuse white point corresponding to some linear *RGB* value *(R, G, B)*, where usually $R = G = B = 1.0$. Note that standard CIE notation uses lowercase letters to specify

chromaticity coordinates and uppercase letters to specify tristimulus values. Given this information, **Matrix** and **WhitePoint** can be found as follows:

$$z = y_W \times ((x_G - x_B) \times y_R - (x_R - x_B) \times y_G + (x_R - x_G) \times y_B)$$

$$Y_A = \frac{y_R}{R} \times \frac{(x_G - x_B) \times y_W - (x_W - x_B) \times y_G + (x_W - x_G) \times y_B}{z}$$

$$X_A = Y_A \times \frac{x_R}{y_R} \qquad Z_A = Y_A \times \left(\frac{1 - x_R}{y_R} - 1 \right)$$

$$Y_B = -\frac{y_G}{G} \times \frac{(x_R - x_B) \times y_W - (x_W - x_B) \times y_R + (x_W - x_R) \times y_B}{z}$$

$$X_B = Y_B \times \frac{x_G}{y_G} \qquad Z_B = Y_B \times \left(\frac{1 - x_G}{y_G} - 1 \right)$$

$$Y_C = \frac{y_B}{B} \times \frac{(x_R - x_G) \times y_W - (x_W - x_G) \times y_R + (x_W - x_R) \times y_G}{z}$$

$$X_C = Y_C \times \frac{x_B}{y_B} \qquad Z_C = Y_C \times \left(\frac{1 - x_B}{y_B} - 1 \right)$$

$$X_W = X_A \times R + X_B \times G + X_C \times B$$
$$Y_W = Y_A \times R + Y_B \times G + Y_C \times B$$
$$Z_W = Z_A \times R + Z_B \times G + Z_C \times B$$

Lab Color Spaces

A **Lab** color space is a CIE-based *ABC* color space with two transformation stages (see Figure 4.14 on page 245). In this type of space, *A*, *B*, and *C* represent the L^*, a^*, and b^* components of a CIE 1976 $L^*a^*b^*$ space. The range of the first (L^*) component is always 0 to 100; the ranges of the second and third (a^* and b^*) components are defined by the **Range** entry in the color space dictionary (see Table 4.15).

Plate 3 illustrates the coordinates of a typical **Lab** color space; Plate 4 compares the gamuts (ranges of representable colors) for $L^*a^*b^*$, *RGB*, and *CMYK* spaces.

TABLE 4.15 Entries in a Lab color space dictionary

KEY	TYPE	VALUE
WhitePoint	array	*(Required)* An array of three numbers $[X_W\ Y_W\ Z_W]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see "CalRGB Color Spaces" on page 247 for further discussion. The numbers X_W and Z_W must be positive, and Y_W must be equal to 1.0.
BlackPoint	array	*(Optional)* An array of three numbers $[X_B\ Y_B\ Z_B]$ specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see "CalRGB Color Spaces" on page 247 for further discussion. All three of these numbers must be non-negative. Default value: [0.0 0.0 0.0].
Range	array	*(Optional)* An array of four numbers $[a_{min}\ a_{max}\ b_{min}\ b_{max}]$ specifying the range of valid values for the a^* and b^* (*B* and *C*) components of the color space—that is, $a_{min} \le a^* \le a_{max}$ and $b_{min} \le b^* \le b_{max}$ Component values falling outside the specified range are adjusted to the nearest valid value without error indication. Default value: [−100 100 −100 100].

A Lab color space does not specify explicit decoding functions or matrix coefficients for either stage of the transformation from $L^*a^*b^*$ space to *XYZ* space (denoted by "Decode *ABC*," "Matrix *ABC*," "Decode *LMN*," and "Matrix *LMN*" in Figure 4.14 on page 245). Instead, these parameters have constant implicit values. The first transformation stage is defined by the equations

$$L = \frac{L^* + 16}{116} + \frac{a^*}{500}$$

$$M = \frac{L^* + 16}{116}$$

$$N = \frac{L^* + 16}{116} - \frac{b^*}{200}$$

The second transformation stage is given by

$$X = X_W \times g(L)$$
$$Y = Y_W \times g(M)$$
$$Z = Z_W \times g(N)$$

where the function $g(x)$ is defined as

$$g(x) = x^3 \qquad \text{if } x \geq \frac{6}{29}$$

$$g(x) = \frac{108}{841} \times \left(x - \frac{4}{29}\right) \qquad \text{otherwise}$$

Example 4.8 defines the CIE 1976 $L^*a^*b^*$ space with the CCIR XA/11–recommended D65 white point. The a^* and b^* components, although theoretically unbounded, are defined to lie in the useful range −128 to +127.

Example 4.8

```
[ /Lab
      << /WhitePoint [0.9505 1.0000 1.0890]
         /Range [-128 127 -128 127]
      >>
]
```

ICCBased Color Spaces

ICCBased color spaces *(PDF 1.3)* are based on a cross-platform *color profile* as defined by the International Color Consortium (ICC). Unlike the **CalGray**, **CalRGB**, and **Lab** color spaces, which are characterized by entries in the color space dictionary, an **ICCBased** color space is characterized by a sequence of bytes in a standard format. Details of the profile format can be found in the ICC specification (see the Bibliography).

An **ICCBased** color space is specified as an array:

```
[/ICCBased stream]
```

The stream contains the ICC profile. Besides the usual entries common to all streams (see Table 3.4 on page 62), the profile stream has the additional entries listed in Table 4.16.

TABLE 4.16 Additional entries specific to an ICC profile stream dictionary

KEY	TYPE	VALUE
N	integer	*(Required)* The number of color components in the color space described by the ICC profile data. This number must match the number of components actually in the ICC profile. As of PDF 1.4, N must be 1, 3, or 4.
Alternate	array or name	*(Optional)* An alternate color space to be used in case the one specified in the stream data is not supported (for example, by applications designed for earlier versions of PDF). The alternate space may be any valid color space (except a Pattern color space) that has the number of components specified by N. If this entry is omitted and the application does not understand the ICC profile data, the color space used is DeviceGray, DeviceRGB, or DeviceCMYK, depending on whether the value of N is 1, 3, or 4, respectively. *Note: There is no conversion of source color values, such as a tint transformation, when using the alternate color space. Color values within the range of the ICCBased color space might not be within the range of the alternate color space. In this case, the nearest values within the range of the alternate space are substituted.*
Range	array	*(Optional)* An array of 2 × N numbers [min_0 max_0 min_1 max_1 ...] specifying the minimum and maximum valid values of the corresponding color components. These values must match the information in the ICC profile. Default value: [0.0 1.0 0.0 1.0 ...].
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* containing metadata for the color space (see Section 10.2.2, "Metadata Streams").

The ICC specification is an evolving standard. Table 4.17 shows the versions of the ICC specification on which the ICCBased color spaces supported by PDF versions 1.3 and later are based. (Earlier versions of the ICC specification are also supported.)

TABLE 4.17 ICC specification versions supported by ICCBased color spaces

PDF VERSION	ICC SPECIFICATION VERSION
1.3	3.3
1.4	ICC.1:1998-09 and its addendum ICC.1A:1999-04
1.5	ICC.1:2001-12
1.6	ICC.1:2003-09

PDF VERSION	ICC SPECIFICATION VERSION
1.7	ICC.1:2004-10

PDF producers and consumers should follow these guidelines:

- A consumer that supports a given PDF version is required to support ICC profiles conforming to the corresponding version (and earlier versions) of the ICC specification, as described above. It may optionally support later ICC versions.
- For the most predictable and consistent results, a producer of a given PDF version should embed only profiles conforming to the corresponding version of the ICC specification.
- A PDF producer may embed profiles conforming to a later ICC version, with the understanding that the results will vary depending on the capabilities of the consumer. The consumer might process the profile while ignoring newer features, or it might fail altogether to process the profile. Therefore, it is recommended that the producer provide an alternate color space (**Alternate** entry in the **ICCBased** color space dictionary) containing a profile that is appropriate for the PDF version.

PDF supports only the profile types shown in Table 4.18; other types may be supported in the future. (In particular, note that *XYZ* and 16-bit $L^*a^*b^*$ profiles are not supported.) Each of the indicated fields must have one of the values listed for that field in the second column of the table. (Profiles must satisfy *both* the criteria shown in the table.) The terminology is taken from the ICC specifications.

TABLE 4.18 ICC profile types

HEADER FIELD	REQUIRED VALUE
deviceClass	icSigInputClass ('scnr') icSigDisplayClass ('mntr') icSigOutputClass ('prtr') icSigColorSpaceClass ('spac')
colorSpace	icSigGrayData ('GRAY') icSigRgbData ('RGB ') icSigCmykData ('CMYK') icSigLabData ('Lab ')

The terminology used in PDF color spaces and ICC color profiles is similar, but sometimes the same terms are used with different meanings. For example, the default value for each component in an **ICCBased** color space is 0. The range of each color component is a function of the color space specified by the profile and is indicated in the ICC specification. The ranges for several ICC color spaces are shown in Table 4.19.

TABLE 4.19 Ranges for typical ICC color spaces

ICC COLOR SPACE	COMPONENT RANGES
Gray	[0.0 1.0]
RGB	[0.0 1.0]
CMYK	[0.0 1.0]
L*a*b*	L^*: [0 100]; a^* and b^*: [−128 127]

Since the **ICCBased** color space is being used as a source color space, only the "to CIE" profile information (*AToB* in ICC terminology) is used; the "from CIE" (*BToA*) information is ignored when present. An ICC profile may also specify a *rendering intent*, but PDF consumer applications ignore this information; the rendering intent is specified in PDF by a separate parameter (see "Rendering Intents" on page 260).

Note: The requirements stated above apply to an **ICCBased** *color space that is used to specify the source colors of graphics objects. When such a space is used as the blending color space for a transparency group in the transparent imaging model (see Sections 7.2.3, "Blending Color Space"; 7.3, "Transparency Groups"; and 7.5.5, "Transparency Group XObjects"), it must have both "to CIE"* (AToB) *and "from CIE"* (BToA) *information. This is because the group color space is used as both the destination for objects being painted within the group and the source for the group's results. ICC profiles are also used in specifying* output intents *for matching the color characteristics of a PDF document with those of a target output device or production environment. When used in this context, they are subject to still other constraints on the "to CIE" and "from CIE" information; see Section 10.10.4, "Output Intents," for details.*

The representations of **ICCBased** color spaces are less compact than **CalGray**, **CalRGB**, and **Lab**, but can represent a wider range of color spaces. In those cases where a given color space can be expressed by more than one of the CIE-based

color space families, the resulting colors are expected to be rendered similarly, regardless of the method selected for representation.

One particular color space is the so-called "standard *RGB*" or *sRGB*, defined in the International Electrotechnical Commission (IEC) document *Colour Measurement and Management in Multimedia Systems and Equipment* (see the Bibliography). In PDF, the *sRGB* color space can be expressed precisely only as an **ICCBased** space, although it can be approximated by a **CalRGB** space.

Example 4.9 shows an **ICCBased** color space for a typical three-component *RGB* space. The profile's data has been encoded in hexadecimal representation for readability; in actual practice, a lossless decompression filter such as **FlateDecode** should be used.

Example 4.9

```
10 0 obj                                    % Color space
   [/ICCBased 15 0 R]
endobj

15 0 obj                                    % ICC profile stream
   << /N 3
      /Alternate /DeviceRGB
      /Length 1605
      /Filter /ASCIIHexDecode
   >>
stream
00 00 02 0C 61 70 70 6C 02 00 00 00 6D 6E 74 72
52 47 42 20 58 59 5A 20 07 CB 00 02 00 16 00 0E
00 22 00 2C 61 63 73 70 41 50 50 4C 00 00 00 00
61 70 70 6C 00 00 04 01 00 00 00 00 00 00 00 02
00 00 00 00 00 00 F6 D4 00 01 00 00 00 00 D3 2B
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 09 64 65 73 63 00 00 00 F0 00 00 00 71
72 58 59 5A 00 00 01 64 00 00 00 14 67 58 59 5A
00 00 01 78 00 00 00 14 62 58 59 5A 00 00 01 8C
00 00 00 14 72 54 52 43 00 00 01 A0 00 00 00 0E
67 54 52 43 00 00 01 B0 00 00 00 0E 62 54 52 43
00 00 01 C0 00 00 00 0E 77 74 70 74 00 00 01 D0
00 00 00 14 63 70 72 74 00 00 01 E4 00 00 00 27
64 65 73 63 00 00 00 00 00 00 00 17 41 70 70 6C
```

```
65 20 31 33 22 20 52 47 42 20 53 74 61 6E 64 61
72 64 00 00 00 00 00 00 00 00 00 00 17 41 70
70 6C 65 20 31 33 22 20 52 47 42 20 53 74 61 6E
64 61 72 64 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 58 59 5A 58 59 5A 20 00 00 00 00 00 00 63 0A
00 00 35 0F 00 00 03 30 58 59 5A 20 00 00 00 00
00 00 53 3D 00 00 AE 37 00 00 15 76 58 59 5A 20
00 00 00 00 00 00 40 89 00 00 1C AF 00 00 BA 82
63 75 72 76 00 00 00 00 00 00 00 01 01 CC 63 75
63 75 72 76 00 00 00 00 00 00 00 01 01 CC 63 75
63 75 72 76 00 00 00 00 00 00 00 01 01 CC 58 59
58 59 5A 20 00 00 00 00 00 00 F3 1B 00 01 00 00
00 01 67 E7 74 65 78 74 00 00 00 00 20 43 6F 70
79 72 69 67 68 74 20 41 70 70 6C 65 20 43 6F 6D
70 75 74 65 72 73 20 31 39 39 34 00 >
endstream
endobj
```

Default Color Spaces

Colors that are specified in a device color space (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**) are device-dependent. By setting *default color spaces (PDF 1.1)*, a PDF document can request that such colors be systematically transformed *(remapped)* into device-independent CIE-based color spaces. This capability can be useful in a variety of circumstances:

- A document originally intended for one output device is redirected to a different device.
- A document is intended to be compatible with applications designed for earlier versions of PDF and thus cannot specify CIE-based colors directly.
- Color corrections or rendering intents need to be applied to device colors (see "Rendering Intents" on page 260).

A color space is selected for painting each graphics object. This is either the current color space parameter in the graphics state or a color space given as an entry in an image XObject, inline image, or shading dictionary. Regardless of how the color space is specified, it may be subject to remapping as described below.

When a device color space is selected, the **ColorSpace** subdictionary of the current resource dictionary (see Section 3.7.2, "Resource Dictionaries") is checked for the presence of an entry designating a corresponding default color space (**DefaultGray**, **DefaultRGB**, or **DefaultCMYK**, corresponding to **DeviceGray**, **DeviceRGB**, or **DeviceCMYK**, respectively). If such an entry is present, its value is used as the color space for the operation currently being performed. (If the application does not recognize this color space, no remapping occurs; the original device color space is used.)

Color values in the original device color space are passed unchanged to the default color space, which must have the same number of components as the original space. The default color space should be chosen to be compatible with the original, taking into account the components' ranges and whether the components are additive or subtractive. If a color value lies outside the range of the default color space, it is adjusted to the nearest valid value.

*Note: Any color space other than a **Lab**, **Indexed**, or **Pattern** color space may be used as a default color space provided that it is compatible with the original device color space as described above.*

If the selected space is a special color space based on an underlying device color space, the default color space is used in place of the underlying space. This applies to the following color spaces:

- The underlying color space of a **Pattern** color space
- The base color space of an **Indexed** color space
- The alternate color space of a **Separation** or **DeviceN** color space (but only if the alternate color space is actually selected)

See Section 4.5.5, "Special Color Spaces," for details on these color spaces.

*Note: There is no conversion of color values, such as a tint transformation, when using the default color space. Color values that are within the range of the device color space might not be within the range of the default color space (particularly if the default is an **ICCBased** color space). In this case, the nearest values within the range of the default space are used. For this reason, a **Lab** color space is not permitted as the **DefaultRGB** color space.*

Implicit Conversion of CIE-Based Color Spaces

In workflows in which PDF documents are intended for rendering on a specific target output device (such as a printing press with particular inks and media), it is often useful to specify the source colors for some or all of a document's objects in a CIE-based color space that matches the calibration of the intended device. The resulting document, although tailored to the specific characteristics of the target device, remains device-independent and will produce reasonable results if retargeted to a different output device. However, the expectation is that if the document is printed on the intended target device, source colors that have been specified in a color space matching the calibration of the device will pass through unchanged, without conversion to and from the intermediate CIE 1931 *XYZ* space as depicted in Figure 4.14 on page 245.

In particular, when colors intended for a *CMYK* output device are specified in an **ICCBased** color space using a matching *CMYK* printing profile, converting such colors from four components to three and back is unnecessary and results in a loss of fidelity in the black component. In such cases, PDF consumer applications may provide the ability for the user to specify a particular calibration to use for printing, proofing, or previewing. This calibration is then considered to be that of the native color space of the intended output device (typically **DeviceCMYK**), and colors expressed in a CIE-based source color space matching it can be treated as if they were specified directly in the device's native color space. Note that the conditions under which such implicit conversion is done cannot be specified in PDF, since nothing in PDF describes the calibration of the output device (although an output intent dictionary, if present, may suggest such a calibration; see Section 10.10.4, "Output Intents"). The conversion is completely hidden by the application and plays no part in the interpretation of PDF color spaces.

When this type of implicit conversion is done, all of the semantics of the device color space should also apply, even though they do not apply to CIE-based spaces in general. In particular:

- The nonzero overprint mode (see Section 4.5.6, "Overprint Control") determines the interpretation of color component values in the space.
- If the space is used as the blending color space for a transparency group in the transparent imaging model (see Sections 7.2.3, "Blending Color Space"; 7.3, "Transparency Groups"; and 7.5.5, "Transparency Group XObjects"), components of the space, such as **Cyan**, can be selected in a **Separation** or **DeviceN** col-

or space used within the group (see "Separation Color Spaces" on page 264 and "DeviceN Color Spaces" on page 268).

- Likewise, any uses of device color spaces for objects within such a transparency group have well-defined conversions to the group color space.

Note: A source color space can be specified directly (for example, with an ***ICCBased*** *color space) or indirectly using the default color space mechanism (for example,* ***DefaultCMYK****; see "Default Color Spaces" on page 257). The implicit conversion of a CIE-based color space to a device space should not depend on whether the CIE-based space is specified directly or indirectly.*

Rendering Intents

Although CIE-based color specifications are theoretically device-independent, they are subject to practical limitations in the color reproduction capabilities of the output device. Such limitations may sometimes require compromises to be made among various properties of a color specification when rendering colors for a given device. Specifying a *rendering intent (PDF 1.1)* allows a PDF file to set priorities regarding which of these properties to preserve and which to sacrifice. For example, the PDF file might request that colors falling within the output device's gamut (the range of colors it can reproduce) be rendered exactly while sacrificing the accuracy of out-of-gamut colors, or that a scanned image such as a photograph be rendered in a perceptually pleasing manner at the cost of strict colorimetric accuracy.

Rendering intents are specified with the **ri** operator (see Section 4.3.3, "Graphics State Operators"), the **RI** entry in a graphics state parameter dictionary (see Section 4.3.4), and with the **Intent** entry in image dictionaries (Section 4.8.4, "Image Dictionaries"). The value is a name identifying the rendering intent. Table 4.20 lists the standard rendering intents recognized in the initial release of PDF viewer applications from Adobe Systems; Plate 5 illustrates their effects. These intents have been deliberately chosen to correspond closely to those defined by the International Color Consortium (ICC), an industry organization that has developed standards for device-independent color. Note, however, that the exact set of rendering intents supported may vary from one output device to another; a particular device may not support all possible intents or may support additional ones beyond those listed in the table. If the application does not recognize the specified name, it uses the **RelativeColorimetric** intent by default.

See Section 7.6.4, "Rendering Parameters and Transparency," and in particular "Rendering Intent and Color Conversions" on page 574, for further discussion of the role of rendering intents in the transparent imaging model.

TABLE 4.20 Rendering Intents

NAME	DESCRIPTION
AbsoluteColorimetric	Colors are represented solely with respect to the light source; no correction is made for the output medium's white point (such as the color of unprinted paper). Thus, for example, a monitor's white point, which is bluish compared to that of a printer's paper, would be reproduced with a blue cast. In-gamut colors are reproduced exactly; out-of-gamut colors are mapped to the nearest value within the reproducible gamut. This style of reproduction has the advantage of providing exact color matches from one output medium to another. It has the disadvantage of causing colors with Y values between the medium's white point and 1.0 to be out of gamut. A typical use might be for logos and solid colors that require exact reproduction across different media.
RelativeColorimetric	Colors are represented with respect to the combination of the light source and the output medium's white point (such as the color of unprinted paper). Thus, for example, a monitor's white point would be reproduced on a printer by simply leaving the paper unmarked, ignoring color differences between the two media. In-gamut colors are reproduced exactly; out-of-gamut colors are mapped to the nearest value within the reproducible gamut. This style of reproduction has the advantage of adapting for the varying white points of different output media. It has the disadvantage of not providing exact color matches from one medium to another. A typical use might be for vector graphics.
Saturation	Colors are represented in a manner that preserves or emphasizes saturation. Reproduction of in-gamut colors may or may not be colorimetrically accurate. A typical use might be for business graphics, where saturation is the most important attribute of the color.

NAME	DESCRIPTION
Perceptual	Colors are represented in a manner that provides a pleasing perceptual appearance. To preserve color relationships, both in-gamut and out-of-gamut colors are generally modified from their precise colorimetric values. A typical use might be for scanned images.

4.5.5 Special Color Spaces

Special color spaces add features or properties to an underlying color space. There are four special color space families: **Pattern**, **Indexed**, **Separation**, and **DeviceN**.

Pattern Color Spaces

A **Pattern** color space *(PDF 1.2)* enables a PDF content stream to paint an area with a *pattern* rather than a single color. The pattern may be either a *tiling pattern* (type 1) or a *shading pattern* (type 2). Section 4.6, "Patterns," discusses patterns in detail.

Indexed Color Spaces

An **Indexed** color space allows a PDF content stream to use small integers as indices into a *color map* or *color table* of arbitrary colors in some other space. A PDF consumer application treats each sample value as an index into the color table and uses the color value it finds there. This technique can considerably reduce the amount of data required to represent a sampled image—for example, by using 8-bit index values as samples instead of 24-bit *RGB* color values.

An **Indexed** color space is defined by a four-element array:

```
[/Indexed base hival lookup]
```

The first element is the color space family name **Indexed**. The remaining elements are parameters that an **Indexed** color space requires; their meanings are discussed below. Setting the current stroking or nonstroking color space to an **Indexed** color space initializes the corresponding current color to 0.

The *base* parameter is an array or name that identifies the *base color space* in which the values in the color table are to be interpreted. It can be any device or CIE-based color space or (in PDF 1.3) a **Separation** or **DeviceN** space, but not a **Pattern** space or another **Indexed** space. For example, if the base color space is **DeviceRGB**, the values in the color table are to be interpreted as red, green, and blue components; if the base color space is a CIE-based *ABC* space such as a **CalRGB** or **Lab** space, the values are to be interpreted as *A*, *B*, and *C* components.

*Note: Attempting to use a **Separation** or **DeviceN** color space as the base for an **Indexed** color space generates an error in PDF 1.2.*

The *hival* parameter is an integer that specifies the maximum valid index value. In other words, the color table is to be indexed by integers in the range 0 to *hival*. *hival* can be no greater than 255, which is the integer required to index a table with 8-bit index values.

The color table is defined by the *lookup* parameter, which can be either a stream or (in PDF 1.2) a byte string. It provides the mapping between index values and the corresponding colors in the base color space.

The color table data must be $m \times (hival + 1)$ bytes long, where m is the number of color components in the base color space. Each byte is an unsigned integer in the range 0 to 255 that is scaled to the range of the corresponding color component in the base color space; that is, 0 corresponds to the minimum value in the range for that component, and 255 corresponds to the maximum.

*Note: PostScript uses a different interpretation of an **Indexed** color space's color table. In PostScript, the component value is always scaled to the range 0.0 to 1.0, regardless of the range of color values in the base color space.*

The color components for each entry in the table appear consecutively in the string or stream. For example, if the base color space is **DeviceRGB** and the indexed color space contains two colors, the order of bytes in the string or stream is $R_0\ G_0\ B_0\ R_1\ G_1\ B_1$, where letters denote the color component and numeric subscripts denote the table entry.

Example 4.10 illustrates the specification of an **Indexed** color space that maps 8-bit index values to three-component color values in the **DeviceRGB** color space.

Example 4.10

```
[ /Indexed
      /DeviceRGB
      255
      <000000 FF0000 00FF00 0000FF B57342 ...>
]
```

The example shows only the first five color values in the *lookup* string; in all, there should be 256 color values and the string should be 768 bytes long. Having established this color space, the program can now specify colors as single-component values in the range 0 to 255. For example, a color value of 4 selects an *RGB* color whose components are coded as the hexadecimal integers B5, 73, and 42. Dividing these by 255 and scaling the results to the range 0.0 to 1.0 yields a color with red, green, and blue components of 0.710, 0.451, and 0.259, respectively.

Although an **Indexed** color space is useful mainly for images, index values can also be used with the color selection operators **SC**, **SCN**, **sc**, and **scn**. For example:

```
123 sc
```

selects the same color as does an image sample value of 123. The index value should be an integer in the range 0 to *hival*. If the value is a real number, it is rounded to the nearest integer; if it is outside the range 0 to *hival*, it is adjusted to the nearest value within that range.

Separation Color Spaces

Color output devices produce full color by combining *primary* or *process colorants* in varying amounts. On an additive color device such as a display, the primary colorants consist of red, green, and blue phosphors; on a subtractive device such as a printer, they typically consist of cyan, magenta, yellow, and sometimes black inks. In addition, some devices can apply special colorants, often called *spot colorants*, to produce effects that cannot be achieved with the standard process colorants alone. Examples include metallic and fluorescent colors and special textures.

When printing a page, most devices produce a single *composite* page on which all process colorants (and spot colorants, if any) are combined. However, some devices, such as imagesetters, produce a separate, monochromatic rendition of the page, called a *separation*, for each colorant. When the separations are later com-

bined—on a printing press, for example—and the proper inks or other colorants are applied to them, the result is a full-color page.

A **Separation** color space *(PDF 1.2)* provides a means for specifying the use of additional colorants or for isolating the control of individual color components of a device color space for a subtractive device. When such a space is the current color space, the current color is a single-component value, called a *tint*, that controls the application of the given colorant or color components only.

Note: The term separation *is often misused as a synonym for an individual device colorant. In the context of this discussion, a printing system that produces separations generates a separate piece of physical medium (generally film) for each colorant. It is these pieces of physical medium that are correctly referred to as separations. A particular colorant properly constitutes a separation only if the device is generating physical separations, one of which corresponds to the given colorant. The* **Separation** *color space is so named for historical reasons, but it has evolved to the broader purpose of controlling the application of individual colorants in general, regardless of whether they are actually realized as physical separations.*

Note also that the operation of a **Separation** *color space itself is independent of the characteristics of any particular output device. Depending on the device, the space may or may not correspond to a true, physical separation or to an actual colorant. For example, a* **Separation** *color space could be used to control the application of a single process colorant (such as cyan) on a composite device that does not produce physical separations, or could represent a color (such as orange) for which no specific colorant exists on the device. A* **Separation** *color space provides consistent, predictable behavior, even on devices that cannot directly generate the requested color.*

A **Separation** color space is defined as follows:

```
[/Separation name alternateSpace tintTransform]
```

In other words, it is a four-element array whose first element is the color space family name **Separation**. The remaining elements are parameters that a **Separation** color space requires; their meanings are discussed below.

A color value in a **Separation** color space consists of a single tint component in the range 0.0 to 1.0. The value 0.0 represents the minimum amount of colorant that can be applied; 1.0 represents the maximum. Tints are always treated as *subtractive* colors, even if the device produces output for the designated component by an additive method. Thus, a tint value of 0.0 denotes the lightest color

that can be achieved with the given colorant, and 1.0 is the darkest. (This convention is the same as for **DeviceCMYK** color components but opposite to the one for **DeviceGray** and **DeviceRGB**.) The initial value for both the stroking and nonstroking color in the graphics state is 1.0. The **SCN** and **scn** operators respectively set the current stroking and nonstroking color to a tint value. A sampled image with single-component samples can also be used as a source of tint values.

The *name* parameter is a name object specifying the name of the colorant that this **Separation** color space is intended to represent (or one of the special names **All** or **None**; see below). Such colorant names are arbitrary, and there can be any number of them, subject to implementation limits.

The special colorant name **All** refers collectively to all colorants available on an output device, including those for the standard process colorants. When a **Separation** space with this colorant name is the current color space, painting operators apply tint values to all available colorants at once. This is useful for purposes such as painting registration targets in the same place on every separation. Such marks are typically painted as the last step in composing a page to ensure that they are not overwritten by subsequent painting operations.

The special colorant name **None** never produces any visible output. Painting operations in a **Separation** space with this colorant name have no effect on the current page.

All devices support **Separation** color spaces with the colorant names **All** and **None**, even if they do not support any others. **Separation** spaces with either of these colorant names ignore the *alternateSpace* and *tintTransform* parameters (discussed below), although valid values must still be provided.

At the moment the color space is set to a **Separation** space, the consumer application determines whether the device has an available colorant corresponding to the name of the requested space. If so, the application ignores the *alternateSpace* and *tintTransform* parameters; subsequent painting operations within the space apply the designated colorant directly, according to the tint values supplied.

Note: The preceding paragraph applies only to subtractive output devices such as printers and imagesetters. For an additive device such as a computer display, a **Separation** *color space never applies a process colorant directly; it always reverts to the alternate color space as described below. This is because the model of applying process colorants independently does not work as intended on an additive device; for*

instance, painting tints of the **Red** *component on a white background produces a result that varies from white to cyan.*

Note that this exception applies only to colorants for additive devices, not to the specific names **Red**, **Green**, *and* **Blue**. *In contrast, a printer might have a (subtractive) ink named, for example,* **Red**, *which should work as a* **Separation** *color space just the same as any other supported colorant.*

If the colorant name associated with a **Separation** color space does not correspond to a colorant available on the device, the application arranges for subsequent painting operations to be performed in an *alternate color space*. The intended colors can be approximated by colors in a device or CIE-based color space, which are then rendered with the usual primary or process colorants:

- The *alternateSpace* parameter must be an array or name object that identifies the alternate color space, which can be any device or CIE-based color space but not another special color space (**Pattern**, **Indexed**, **Separation**, or **DeviceN**).
- The *tintTransform* parameter must be a function (see Section 3.9, "Functions"). During subsequent painting operations, an application calls this function to transform a tint value into color component values in the alternate color space. The function is called with the tint value and must return the corresponding color component values. That is, the number of components and the interpretation of their values depend on the alternate color space.

Note: *Painting in the alternate color space may produce a good approximation of the intended color when only opaque objects are painted. However, it does not correctly represent the interactions between an object and its backdrop when the object is painted with transparency or when overprinting (see Section 4.5.6, "Overprint Control") is enabled.*

Example 4.11 illustrates the specification of a **Separation** color space (object 5) that is intended to produce a color named LogoGreen. If the output device has no colorant corresponding to this color, **DeviceCMYK** is used as the alternate color space, and the tint transformation function (object 12) maps tint values linearly into shades of a *CMYK* color value approximating the LogoGreen color.

Example 4.11

```
5 0 obj                                          % Color space
   [ /Separation
         /LogoGreen
         /DeviceCMYK
         12 0 R
   ]
endobj

12 0 obj                                         % Tint transformation function
   << /FunctionType 4
      /Domain [0.0 1.0]
      /Range [0.0 1.0  0.0 1.0  0.0 1.0  0.0 1.0]
      /Length 62
   >>
stream
   { dup 0.84 mul
     exch 0.00 exch dup 0.44 mul
     exch 0.21 mul
   }
endstream
endobj
```

See Section 7.6.2, "Spot Colors and Transparency," for further discussion of the role of **Separation** color spaces in the transparent imaging model.

DeviceN Color Spaces

DeviceN color spaces *(PDF 1.3)* can contain an arbitrary number of color components. They provide greater flexibility than is possible with standard device color spaces such as **DeviceCMYK** or with individual **Separation** color spaces. For example, it is possible to create a **DeviceN** color space consisting of only the cyan, magenta, and yellow color components, with the black component excluded.

DeviceN color spaces are used in applications such as these:

- *High-fidelity* color is the use of more than the standard *CMYK* process colorants to produce an extended *gamut*, or range of colors. A popular example is

the PANTONE Hexachrome system, which uses six colorants: the usual cyan, magenta, yellow, and black, plus orange and green.

- *Multitone* color systems use a single-component image to specify multiple color components. In a *duotone*, for example, a single-component image can be used to specify both the black component and a spot color component. The tone reproduction is generally different for the different components. For example, the black component might be painted with the exact sample data from the single-component image; the spot color component might be generated as a nonlinear function of the image data in a manner that emphasizes the shadows. Plate 6 shows an example that uses black and magenta color components. In Plate 7, a single-component grayscale image is used to generate a *quadtone* result that uses four colorants: black and three PANTONE spot colors. See Example 4.21 on page 282 for the code used to generate this image.

DeviceN was designed to represent color spaces containing multiple components that correspond to colorants of some target device. As with **Separation** color spaces, PDF consumer applications must be able to approximate the colorants if they are not available on the current output device, such as a display. To accomplish this, the color space definition provides a tint transformation function that can be used to convert all the components to an alternate color space.

PDF 1.6 extends the meaning of **DeviceN** to include color spaces that are referred to as ***NChannel color spaces.*** Such color spaces may contain an arbitrary number of spot and process components, which may or may not correspond to specific device colorants (the process components must be from a single process color space). They provide information about each component that allows applications more flexibility in converting colors. For example, they may use their own blending algorithms for on-screen viewing and composite printing, rather than being required to use a specified tint transformation function. These color spaces are identified by a value of **NChannel** for the **Subtype** entry of the attributes dictionary (see Table 4.21). A value of **DeviceN** for the **Subtype** entry, or no value, means that only the previous features are supported. PDF consumer applications that do not support PDF 1.6 treat these color spaces as normal **DeviceN** color spaces and use the tint transformation function as appropriate. Producer applications using the **NChannel** features should follow certain guidelines, as noted throughout this section, to achieve good backward compatibility.

DeviceN color spaces are defined in a similar way to **Separation** color spaces—in fact, a **Separation** color space can be defined as a **DeviceN** color space with only one component.

A **DeviceN** color space is specified as follows:

[/DeviceN *names alternateSpace tintTransform*]

or

[/DeviceN *names alternateSpace tintTransform attributes*]

It is a four- or five-element array whose first element is the color space family name **DeviceN**. The remaining elements are parameters that a **DeviceN** color space requires.

The *names* parameter is an array of name objects specifying the individual color components. The length of the array determines the number of components in the **DeviceN** color space, which is subject to an implementation limit; see Appendix C.The component names must all be different from one another, except for the name **None**, which can be repeated as described later in this section. (The special name **All**, used by **Separation** color spaces, is not allowed.)

Color values are tint components in the range 0.0 to 1.0:

- For **DeviceN** color spaces that do not have a subtype of **NChannel**, 0.0 always represents the minimum amount of colorant; 1.0 represents the maximum. Tints are always treated as subtractive colors, even if the device produces output for the designated component by an additive method. Thus, a tint value of 0.0 denotes the lightest color that can be achieved with the given colorant, and 1.0 the darkest. (This convention is the same one as for **DeviceCMYK** color components but opposite to the one for **DeviceGray** and **DeviceRGB**.)
- For **NChannel** color spaces, values for additive process colors (such as *RGB*) are specified in their natural form, where 1.0 represents maximum intensity of color.

When this space is set to the current color space (using the **CS** or **cs** operators), each component is given an initial value of 1.0. The **SCN** and **scn** operators respectively set the current stroking and nonstroking color. Operand values supplied to **SCN** or **scn** are interpreted as color component values in the order in which the colors are given in the *names* array, as are the values in a sampled image that uses a **DeviceN** color space.

The *alternateSpace* parameter is an array or name object that can be any device or CIE-based color space but not another special color space (**Pattern**, **Indexed**, **Separation**, or **DeviceN**). When the color space is set to a **DeviceN** space, if any of

the component names in the color space do not correspond to a colorant available on the device, the PDF consumer application can perform subsequent painting operations in the alternate color space specified by this parameter.

Note: For **NChannel** *color spaces, the components are evaluated individually; that is, only the ones not present on the output device use the alternate color space.*

The *tintTransform* parameter specifies a function (see Section 3.9, "Functions") that is used to transform the tint values into the alternate color space. It is called with *n* tint values and returns *m* color component values, where *n* is the number of components needed to specify a color in the **DeviceN** color space and *m* is the number required by the alternate color space.

Note: Painting in the alternate color space may produce a good approximation of the intended color when only opaque objects are painted. However, it does not correctly represent the interactions between an object and its backdrop when the object is painted with transparency or when overprinting (see Section 4.5.6, "Overprint Control") is enabled.

The color component name **None**, which may be present only for **DeviceN** color spaces that do *not* have the **NChannel** subtype, indicates that the corresponding color component is never painted on the page, as in a **Separation** color space for the **None** colorant. (However, see implementation note 48 in Appendix H.) When a **DeviceN** color space is painting the named device colorants directly, color components corresponding to **None** colorants are discarded. However, when the **DeviceN** color space reverts to its alternate color space, those components are passed to the tint transformation function, which can use them as desired.

Note: A **DeviceN** *color space whose component colorant names are all* **None** *always discards its output, just the same as a* **Separation** *color space for* **None***; it never reverts to the alternate color space. Reversion occurs only if at least one color component (other than* **None***) is specified and is not available on the device.*

The optional *attributes* parameter is a dictionary (see Table 4.21) containing additional information about the components of color space that PDF consumer applications may use. PDF consumers are not required to use the *alternateSpace* and *tintTransform* parameters, and may instead use custom blending algorithms, along with other information provided in the attributes dictionary if present. (If the value of the **Subtype** entry in the attributes dictionary is **NChannel**, such information must be present.) However, *alternateSpace* and *tintTransform* must always be provided for applications that want to use them or do not support PDF 1.6.

TABLE 4.21 Entries in a DeviceN color space attributes dictionary

KEY	TYPE	VALUE
Subtype	name	*(Optional; PDF 1.6)* A name specifying the preferred treatment for the color space. Possible values are **DeviceN** and **NChannel**. Default value: **DeviceN**.
Colorants	dictionary	*(Required if **Subtype** is **NChannel** and the color space includes spot colorants; otherwise optional)* A dictionary describing the individual colorants used in the **DeviceN** color space. For each entry in this dictionary, the key is a colorant name and the value is an array defining a **Separation** color space for that colorant (see "Separation Color Spaces" on page 264). The key must match the colorant name given in that color space. This dictionary provides information about the individual colorants that may be useful to some applications. In particular, the alternate color space and tint transformation function of a **Separation** color space describe the appearance of that colorant alone, whereas those of a **DeviceN** color space describe only the appearance of its colorants in combination. If **Subtype** is **NChannel**, this dictionary must have entries for all spot colorants in this color space. This dictionary may also include additional colorants not used by this color space.
Process	dictionary	*(Required if **Subtype** is **NChannel** and the color space includes components of a process color space, otherwise optional; PDF 1.6)* A dictionary (see Table 4.22) that describes the process color space whose components are included in this color space.
MixingHints	dictionary	*(Optional; PDF 1.6)* A dictionary (see Table 4.23) that specifies optional attributes of the inks to be used in blending calculations when used as an alternative to the tint transformation function.

A value of **NChannel** for the **Subtype** entry indicates that some of the other entries in this dictionary are required rather than optional. The **Colorants** entry specifies a *colorants dictionary* that contains entries for all the spot colorants in the color space; they are defined using individual **Separation** color spaces. The **Process** entry specifies a *process dictionary* (see Table 4.22) that identifies the process color space that is used by this color space and the names of its components. It must be present if **Subtype** is **NChannel** and the color space has process color components. (An **NChannel** color space may contain components from at most one process color space.)

For color spaces that have a value of **NChannel** for the **Subtype** entry in the attributes dictionary (see Table 4.21), the following restrictions apply to process colors:

- There can be color components from at most one process color space, which can be any device or CIE-based color space.
- For a non-*CMYK* color space, the names of the process components must appear sequentially in the *names* array, in the normal color space order (for example, **Red**, **Green**, and **Blue**). However, the names in the *names* array need not match the actual color space names (for example, a **Red** component need not be named **Red**).The mapping of names is specified in the process dictionary (see Table 4.22 and discussion below), which is required to be present.
- Definitions for process colorants should not appear in the colorants dictionary. Any such definition should be ignored if the colorant is also present in the process dictionary. Any component not specified in the process dictionary is considered to be a spot colorant.
- For a *CMYK* color space, a subset of the components may be present, and they may appear in any order in the *names* array. The reserved names **Cyan**, **Magenta**, **Yellow**, and **Black** are always considered to be process colors, which do not necessarily correspond to the colorants of a specific device; they are not required to have entries in the process dictionary.
- The values associated with the process components must be stored in their natural form (that is, subtractive color values for *CMYK* and additive color values for *RGB*), since they are interpreted directly as process values by consumers making use of the process dictionary. (For additive color spaces, this is the reverse of how color values are specified for **DeviceN**, as described above in the discussion of the *names* parameter.)

The **MixingHints** entry in the attributes dictionary specifies a *mixing hints dictionary* (see Table 4.23) that provides information about the characteristics of colorants that can be used in blending calculations when the actual colorants are not available on the target device. Applications are not required to use this information.

TABLE 4.22 Entries in a DeviceN process dictionary

KEY	TYPE	VALUE
ColorSpace	name or array	*(Required)* A name or array identifying the process color space, which may be any device or CIE-based color space. If an **ICCBased** color space is specified, it must provide calibration information appropriate for the process color components specified in the *names* array of the **DeviceN** color space.
Components	array	*(Required)* An array of component names that correspond, in order, to the components of the process color space specified in **ColorSpace**. For example, an RGB color space must have three names corresponding to red, green, and blue. The names may be arbitrary (that is, not the same as the standard names for the color space components) and must match those specified in the *names* array of the **DeviceN** color space, even if all components are not present in the *names* array.

TABLE 4.23 Entries in a DeviceN mixing hints dictionary

KEY	TYPE	VALUE
Solidities	dictionary	*(Optional)* A dictionary specifying the solidity of inks to be used in blending calculations when used as an alternative to the tint transformation function. For each entry, the key is a colorant name, and the value is a number between 0.0 and 1.0. This dictionary need not contain entries for all colorants used in this color space; it may also include additional colorants not used by this color space. A value of 1.0 simulates an ink that completely covers the inks beneath; a value of 0.0 simulates a transparent ink that completely reveals the inks beneath. An entry with a key of **Default** specifies a value to be used by all components in the associated **DeviceN** color space for which a solidity value is not explicitly provided. If **Default** is not present, the default value for unspecified colorants is 0.0; applications may choose to use other values. If this entry is present, **PrintingOrder** must also be present.
PrintingOrder	array	*(Required if Solidities is present)* An array of colorant names, specifying the order in which inks are laid down. Each component in the *names* array of the **DeviceN** color space must appear in this array (although the order is unrelated to the order specified in the *names* array). This entry may also list colorants unused by this specific **DeviceN** instance.

KEY	TYPE	VALUE
DotGain	dictionary	*(Optional)* A dictionary specifying the *dot gain* of inks to be used in blending calculations when used as an alternative to the tint transformation function. Dot gain (or loss) represents the amount by which a printer's halftone dots change as the ink spreads and is absorbed by paper. For each entry, the key is a colorant name, and the value is a function that maps values in the range 0 to 1 to values in the range 0 to 1. The dictionary may list colorants unused by this specific **DeviceN** instance and need not list all colorants. An entry with a key of **Default** specifies a function to be used by all colorants for which a dot gain function is not explicitly specified. PDF consumer applications may ignore values in this dictionary when other sources of dot gain information are available, such as ICC profiles associated with the process color space or tint transformation functions associated with individual colorants.

Each entry in the mixing hints dictionary refers to colorant names, which include spot colorants referenced by the **Colorants** dictionary. Under some circumstances, they may also refer to one or more individual process components called **Cyan**, **Magenta**, **Yellow**, or **Black** when **DeviceCMYK** is specified as the process color space in the process dictionary. However, applications should ignore these process component entries if they can obtain the information from an ICC profile.

Note: The mixing hints subdictionaries (as well as the colorants dictionary) may specify colorants that are not used in any given instance of a **DeviceN** *color space. This allows them to be referenced from multiple* **DeviceN** *color spaces, which can produce smaller file sizes as well as consistent color definitions across instances.*

For consistency of color, PDF consumers should follow these guidelines:

- The consumer should apply either the specified tint transformation function or invoke the same alternative blending algorithm for all **DeviceN** instances in the document.

  Note: When the tint transformation function is used, the burden is on the producer to guarantee that the individual function definitions chosen for all **DeviceN** *instances produce similar color appearances throughout the document.*

- Blending algorithms should produce a similar appearance for colors when they are used as separation colors or as a component of a **DeviceN** color space.

Example 4.12 shows a **DeviceN** color space consisting of three color components named **Orange**, **Green**, and **None**. In this example, the **DeviceN** color space, object 30, has an attributes dictionary whose **Colorants** entry is an indirect reference to object 45 (which might also be referenced by attributes dictionaries of other **DeviceN** color spaces). *tintTransform1*, whose definition is not shown, maps three color components (tints of the colorants **Orange**, **Green**, and **None**) to four color components in the alternate color space, **DeviceCMYK**. *tintTransform2* maps a single color component (an orange tint) to four components in **DeviceCMYK**. Likewise, *tintTransform3* maps a green tint to **DeviceCMYK**, and *tintTransform4* maps a tint of PANTONE 131 to **DeviceCMYK**.

Example 4.12

```
30 0 obj                                          % Color space
   [ /DeviceN
        [/Orange /Green /None]
        /DeviceCMYK
        tintTransform1
        << /Colorants 45 0 R >>
   ]
endobj

45 0 obj                                          % Colorants dictionary
   << /Orange [ /Separation
                   /Orange
                   /DeviceCMYK
                   tintTransform2
              ]
      /Green [ /Separation
                   /Green
                   /DeviceCMYK
                   tintTransform3
             ]
      /PANTONE#20131 [ /Separation
                          /PANTONE#20131
                          /DeviceCMYK
                          tintTransform4
                     ]
   >>
endobj
```

Examples 4.13 through 4.16 show the use of **NChannel** color spaces. Example 4.13 shows the use of calibrated *CMYK* process components. Example 4.14 shows the use of **Lab** process components.

Example 4.13

```
10 0 obj                                          % Color space
  [ /DeviceN
      [/Magenta /Spot1 /Yellow /Spot2]
      alternateSpace
      tintTransform1
      <<                                          % Attributes dictionary
        /Subtype /NChannel
        /Process
          << /ColorSpace [/ICCBased CMYK_ICC profile]
             /Components [/Cyan /Magenta /Yellow /Black]
          >>
        /Colorants
          << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2]
             /Spot2 [/Separation /Spot2 alternateSpace tintTransform3]
          >>
      >>
  ]
endobj
```

Example 4.14

```
10 0 obj                                          % Color space
  [ /DeviceN
    [/L /a /b /Spot1 /Spot2]
    alternateSpace
    tintTransform1
    <<                                            % Attributes dictionary
      /Subtype /NChannel
      /Process
        << /ColorSpace [ /Lab << /WhitePoint ... /Range ... >> ]
           /Components [/L /a /b]
        >>
      /Colorants
        << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2 ]
           /Spot2 [/Separation /Spot2 alternateSpace tintTransform3]
        >>
    >>
  ]
```

Example 4.15 shows the recommended convention for dealing with situations where a spot colorant and a process color component have the same name. Since the *names* array may not have duplicate names, the process colors should be given different names, which are mapped to process components in the **Components** entry of the process dictionary. In this case, **Red** refers to a spot colorant; **ProcessRed**, **ProcessGreen**, and **ProcessBlue** are mapped to the components of an *RGB* color space.

Example 4.15

```
10 0 obj                                              % Color space
   [ /DeviceN
      [/ProcessRed /ProcessGreen /ProcessBlue /Red]
      alternateSpace
      tintTransform1
      <<                                                % Attributes dictionary
         /Subtype /NChannel
         /Process
            << /ColorSpace [ /ICCBased RGB_ICC profile ]
               /Components [/ProcessRed /ProcessGreen /ProcessBlue]
            >>
         /Colorants
            << /Red [/Separation /Red alternateSpace tintTransform2 ] >>
      >>
   ]
```

Example 4.16 shows the use of a mixing hints dictionary.

Example 4.16

```
10 0 obj                                              % Color space
   [/DeviceN
      [/Magenta /Spot1 /Yellow /Spot2]
      alternateSpace
      tintTransform1
      <<
         /Subtype /NChannel
         /Process
            << /ColorSpace [ /ICCBased CMYK_ICC profile ]
               /Components [/Cyan /Magenta /Yellow /Black]
            >>
         /Colorants
            << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2 ]
               /Spot2 [/Separation /Spot2 alternateSpace tintTransform2 ]
```

```
        >>
    /MixingHints
        <<
            /Solidities
                << /Spot1 1.0
                    /Spot2 0.0
                >>
            /DotGain
                << /Spot1 function1
                    /Spot2 function2
                    /Magenta function3
                    /Yellow function4
                >>
            /PrintingOrder [/Magenta /Yellow /Spot1 /Spot2]
        >>
>>
]
```

See Section 7.6.2, "Spot Colors and Transparency," for further discussion of the role of **DeviceN** color spaces in the transparent imaging model.

Multitone Examples

The following examples illustrate various interesting and useful special cases of the use of **Indexed** and **DeviceN** color spaces in combination to produce multitone colors.

Examples 4.17 and 4.18 illustrate the use of **DeviceN** to create duotone color spaces. In Example 4.17, an **Indexed** color space maps index values in the range 0 to 255 to a duotone **DeviceN** space in cyan and black. In effect, the index values are treated as if they were tints of the duotone space, which are then mapped into tints of the two underlying colorants. Only the beginning of the lookup table string for the **Indexed** color space is shown; the full table would contain 256 two-byte entries, each specifying a tint value for cyan and black, for a total of 512 bytes. If the alternate color space of the **DeviceN** space is selected, the tint transformation function (object 15 in the example) maps the two tint components for cyan and black to the four components for a **DeviceCMYK** color space by supplying zero values for the other two components. Example 4.18 shows the definition of another duotone color space, this time using black and gold colorants (where gold is a spot colorant) and using a **CalRGB** space as the alternate color space. This could be defined in the same way as in the preceding example, with a tint trans-

formation function that converts from the two tint components to colors in the alternate **CalRGB** color space.

Example 4.17

```
10 0 obj                                          % Color space
   [ /Indexed
        [ /DeviceN
             [/Cyan /Black]
             /DeviceCMYK
             15 0 R
        ]
        255
        <6605 6806 6907 6B09 6C0A ...>
   ]
endobj

15 0 obj                                          % Tint transformation function
   << /FunctionType 4
      /Domain [0.0 1.0 0.0 1.0]
      /Range [0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0]
      /Length 16
   >>
stream
   {0 0 3 -1 roll}
endstream
endobj
```

Example 4.18

```
30 0 obj                                          % Color space
   [ /Indexed
        [ /DeviceN
             [/Black /Gold]
             [ /CalRGB
                  << /WhitePoint [1.0 1.0 1.0]
                     /Gamma [2.2 2.2 2.2]
                  >>
             ]
             35 0 R                               % Tint transformation function
        ]
        255
        ... Lookup table...
   ]
endobj
```

Given a formula for converting any combination of black and gold tints to calibrated *RGB*, a 2-in, 3-out type 4 (PostScript calculator) function could be used for the tint transformation. Alternatively, a type 0 (sampled) function could be used, but this would require a large number of sample points to represent the function accurately; for example, sampling each input variable for 256 tint values between 0.0 and 1.0 would require $256^2 = 65{,}536$ samples. But since the **DeviceN** color space is being used as the base of an **Indexed** color space, there are actually only 256 possible combinations of black and gold tint values. A more compact way to represent this information is to put the alternate color values directly into the lookup table alongside the **DeviceN** color values, as in Example 4.19.

Example 4.19

```
10 0 obj                                          % Color space
   [ /Indexed
        [ /DeviceN
             [/Black /Gold /None /None /None]
             [  /CalRGB
                  << /WhitePoint [1.0 1.0 1.0]
                     /Gamma [2.2 2.2 2.2]
                  >>
             ]
             20 0 R                               % Tint transformation function
        ]
        255
        ...Lookup table...
   ]
endobj
```

In this example, each entry in the lookup table has *five* components: two for the black and gold colorants and three more (specified as **None**) for the equivalent **CalRGB** color components. If the black and gold colorants are available on the output device, the **None** components are ignored; if black and gold are not available, the tint transformation function is used to convert a five-component color into a three-component equivalent in the alternate **CalRGB** color space. But because, by construction, the third, fourth, and fifth components *are* the **CalRGB** components, the tint transformation function can merely discard the first two components and return the last three. This can be easily expressed with a type 4 (PostScript calculator) function, as shown in Example 4.20.

Example 4.20

```
20 0 obj                                          % Tint transformation function
   << /FunctionType 4
      /Domain [0.0 1.0  0.0 1.0  0.0 1.0  0.0 1.0  0.0 1.0]
      /Range [0.0 1.0  0.0 1.0  0.0 1.0]
      /Length 27
   >>
stream
   {5 3 roll pop pop}
endstream
endobj
```

Example 4.21 uses an extension of the techniques described above to produce the quadtone (four-component) image shown in Plate 7.

Example 4.21

```
5 0 obj                                   % Image XObject
   << /Type /XObject
      /Subtype /Image
      /Width 288
      /Height 288
      /ColorSpace 10 0 R
      /BitsPerComponent 8
      /Length 105278
      /Filter /ASCII85Decode
   >>
stream
...Data for grayscale image...
endstream
endobj

10 0 obj                                  % Indexed color space for image
   [ /Indexed
        15 0 R                            % Base color space
        255                               % Table has 256 entries
        30 0 R                            % Lookup table
   ]
endobj
```

```
15 0 obj                                % Base color space (DeviceN) for Indexed space
   [ /DeviceN
         [ /Black                       % Four colorants (black plus three spot colors)
           /PANTONE#20216#20CVC
           /PANTONE#20409#20CVC
           /PANTONE#202985#20CVC
           /None                        % Three components for alternate space
           /None
           /None
         ]
         16 0 R                         % Alternate color space
         20 0 R                         % Tint transformation function
   ]
endobj

16 0 obj                                % Alternate color space for DeviceN space
   [ /CalRGB
         << /WhitePoint [1.0 1.0 1.0] >>
   ]
endobj

20 0 obj                                % Tint transformation function for DeviceN space
   << /FunctionType 4
      /Domain [0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0]
      /Range [0.0 1.0 0.0 1.0 0.0 1.0]
      /Length 44
   >>
stream
   { 7 3 roll                           % Just discard first four values
     pop pop pop pop
   }
endstream
endobj

30 0 obj                                % Lookup table for Indexed color space
   << /Length 1975
      /Filter [/ASCII85Decode /FlateDecode]
   >>
stream
8;T1BB2"M7*!"psYBt1k\gY1T<D&tO]r*F7Hga*
... Additional data (seven components for each table entry) ...
endstream
endobj
```

As in the preceding examples, an **Indexed** color space based on a **DeviceN** space is used to paint the grayscale image shown on the left in the plate with four colorants: black and three PANTONE spot colors. The alternate color space is a simple calibrated *RGB*. Thus, the **DeviceN** color space has seven components: the four desired colorants plus the three components of the alternate space. The example shows the image XObject (see Section 4.8.4, "Image Dictionaries") representing the quadtone image, followed by the color space used to interpret the image data. (See implementation note 49 in Appendix H.)

4.5.6 Overprint Control

The graphics state contains an *overprint parameter*, controlled by the **OP** and **op** entries in a graphics state parameter dictionary. Overprint control is useful mainly on devices that produce true physical separations, but it is available on some composite devices as well. Although the operation of this parameter is device-dependent, it is described here rather than in the chapter on color rendering, because it pertains to an aspect of painting in device color spaces that is important to many applications.

Any painting operation marks some specific set of device colorants, depending on the color space in which the painting takes place. In a **Separation** or **DeviceN** color space, the colorants to be marked are specified explicitly; in a device or CIE-based color space, they are implied by the process color model of the output device (see Chapter 6). The overprint parameter is a boolean flag that determines how painting operations affect colorants other than those explicitly or implicitly specified by the current color space.

If the overprint parameter is **false** (the default value), painting a color in any color space causes the corresponding areas of unspecified colorants to be erased (painted with a tint value of 0.0). The effect is that the color at any position on the page is whatever was painted there last, which is consistent with the normal painting behavior of the opaque imaging model.

If the overprint parameter is **true** and the output device supports overprinting, no such erasing actions are performed; anything previously painted in other colorants is left undisturbed. Consequently, the color at a given position on the page may be a combined result of several painting operations in different colorants. The effect produced by such overprinting is device-dependent and is not defined by the PDF language.

Note: Not all devices support overprinting. Furthermore, many PostScript printers support it only when separations are being produced, and not for composite output. If overprinting is not supported, the value of the overprint parameter is ignored.

An additional graphics state parameter, the *overprint mode (PDF 1.3)*, affects the interpretation of a tint value of 0.0 for a color component in a **DeviceCMYK** color space when overprinting is enabled. This parameter is controlled by the **OPM** entry in a graphics state parameter dictionary; it has an effect only when the overprint parameter is **true**, as described above.

When colors are specified in a **DeviceCMYK** color space and the native color space of the output device is also **DeviceCMYK**, each of the source color components controls the corresponding device colorant directly. Ordinarily, each source color component value replaces the value previously painted for the corresponding device colorant, no matter what the new value is; this is the default behavior, specified by overprint mode 0.

When the overprint mode is 1 (also called *nonzero overprint mode*), a tint value of 0.0 for a source color component leaves the corresponding component of the previously painted color unchanged. The effect is equivalent to painting in a **DeviceN** color space that includes only those components whose values are nonzero. For example, if the overprint parameter is **true** and the overprint mode is 1, the operation

```
0.2 0.3 0.0 1.0 k
```

is equivalent to

```
0.2 0.3 1.0 scn
```

in the color space shown in Example 4.22.

Example 4.22

```
10 0 obj                                        % Color space
   [ /DeviceN
        [/Cyan /Magenta /Black]
        /DeviceCMYK
        15 0 R
   ]
endobj
```

```
15 0 obj                                          % Tint transformation function
   << /FunctionType 4
      /Domain [0.0 1.0  0.0 1.0  0.0 1.0]
      /Range [0.0 1.0  0.0 1.0  0.0 1.0  0.0 1.0]
      /Length 13
   >>
stream
   {0 exch}
endstream
endobj
```

Nonzero overprint mode applies only to painting operations that use the current color in the graphics state when the current color space is **DeviceCMYK** (or is implicitly converted to **DeviceCMYK**; see "Implicit Conversion of CIE-Based Color Spaces" on page 259). It does not apply to the painting of images or to any colors that are the result of a computation, such as those in a shading pattern or conversions from some other color space. It also does not apply if the device's native color space is not **DeviceCMYK**; in that case, source colors must be converted to the device's native color space, and all components participate in the conversion, whatever their values. (This is shown explicitly in the alternate color space and tint transformation function of the **DeviceN** color space in Example 4.22.)

See Section 7.6.3, "Overprinting and Transparency," for further discussion of the role of overprinting in the transparent imaging model.

4.5.7 Color Operators

Table 4.24 lists the PDF operators that control color spaces and color values. (Also color-related is the graphics state operator **ri**, listed in Table 4.7 on page 219 and discussed under "Rendering Intents" on page 260.) Color operators may appear at the page description level or inside text objects (see Figure 4.1 on page 197).

TABLE 4.24 Color operators

OPERANDS	OPERATOR	DESCRIPTION
name	CS	*(PDF 1.1)* Set the current color space to use for stroking operations. The operand *name* must be a name object. If the color space is one that can be specified by a name and no additional parameters (**DeviceGray**, **DeviceRGB**, **DeviceCMYK**, and certain cases of **Pattern**), the name may be specified directly. Otherwise, it must be a name defined in the **ColorSpace** subdictionary of the current resource dictionary (see Section 3.7.2, "Resource Dictionaries"); the associated value is an array describing the color space (see Section 4.5.2, "Color Space Families"). *Note: The names **DeviceGray**, **DeviceRGB**, **DeviceCMYK**, and **Pattern** always identify the corresponding color spaces directly; they never refer to resources in the **ColorSpace** subdictionary.* The **CS** operator also sets the current stroking color to its initial value, which depends on the color space: • In a **DeviceGray**, **DeviceRGB**, **CalGray**, or **CalRGB** color space, the initial color has all components equal to 0.0. • In a **DeviceCMYK** color space, the initial color is [0.0 0.0 0.0 1.0]. • In a **Lab** or **ICCBased** color space, the initial color has all components equal to 0.0 unless that falls outside the intervals specified by the space's **Range** entry, in which case the nearest valid value is substituted. • In an **Indexed** color space, the initial color value is 0. • In a **Separation** or **DeviceN** color space, the initial tint value is 1.0 for all colorants. • In a **Pattern** color space, the initial color is a pattern object that causes nothing to be painted.
name	cs	*(PDF 1.1)* Same as **CS** but used for nonstroking operations.
$c_1 \ldots c_n$	SC	*(PDF 1.1)* Set the color to use for stroking operations in a device, CIE-based (other than **ICCBased**), or **Indexed** color space. The number of operands required and their interpretation depends on the current stroking color space: • For **DeviceGray**, **CalGray**, and **Indexed** color spaces, one operand is required ($n = 1$). • For **DeviceRGB**, **CalRGB**, and **Lab** color spaces, three operands are required ($n = 3$). • For **DeviceCMYK**, four operands are required ($n = 4$).

OPERANDS	OPERATOR	DESCRIPTION
$c_1 \ldots c_n$ $c_1 \ldots c_n$ *name*	SCN SCN	*(PDF 1.2)* Same as SC but also supports **Pattern**, **Separation**, **DeviceN**, and **ICCBased** color spaces. If the current stroking color space is a **Separation**, **DeviceN**, or **ICCBased** color space, the operands $c_1 \ldots c_n$ are numbers. The number of operands and their interpretation depends on the color space. If the current stroking color space is a **Pattern** color space, *name* is the name of an entry in the **Pattern** subdictionary of the current resource dictionary (see Section 3.7.2, "Resource Dictionaries"). For an uncolored tiling pattern (**PatternType** = 1 and **PaintType** = 2), $c_1 \ldots c_n$ are component values specifying a color in the pattern's underlying color space. For other types of patterns, these operands must not be specified.
$c_1 \ldots c_n$	sc	*(PDF 1.1)* Same as SC but used for nonstroking operations.
$c_1 \ldots c_n$ $c_1 \ldots c_n$ *name*	scn scn	*(PDF 1.2)* Same as SCN but used for nonstroking operations.
gray	G	Set the stroking color space to **DeviceGray** (or the **DefaultGray** color space; see "Default Color Spaces" on page 257) and set the gray level to use for stroking operations. *gray* is a number between 0.0 (black) and 1.0 (white).
gray	g	Same as G but used for nonstroking operations.
r g b	RG	Set the stroking color space to **DeviceRGB** (or the **DefaultRGB** color space; see "Default Color Spaces" on page 257) and set the color to use for stroking operations. Each operand must be a number between 0.0 (minimum intensity) and 1.0 (maximum intensity).
r g b	rg	Same as RG but used for nonstroking operations.
c m y k	K	Set the stroking color space to **DeviceCMYK** (or the **DefaultCMYK** color space; see "Default Color Spaces" on page 257) and set the color to use for stroking operations. Each operand must be a number between 0.0 (zero concentration) and 1.0 (maximum concentration). The behavior of this operator is affected by the overprint mode (see Section 4.5.6, "Overprint Control").
c m y k	k	Same as K but used for nonstroking operations.

Invoking operators that specify colors or other color-related parameters in the graphics state is restricted in certain circumstances. This restriction occurs when