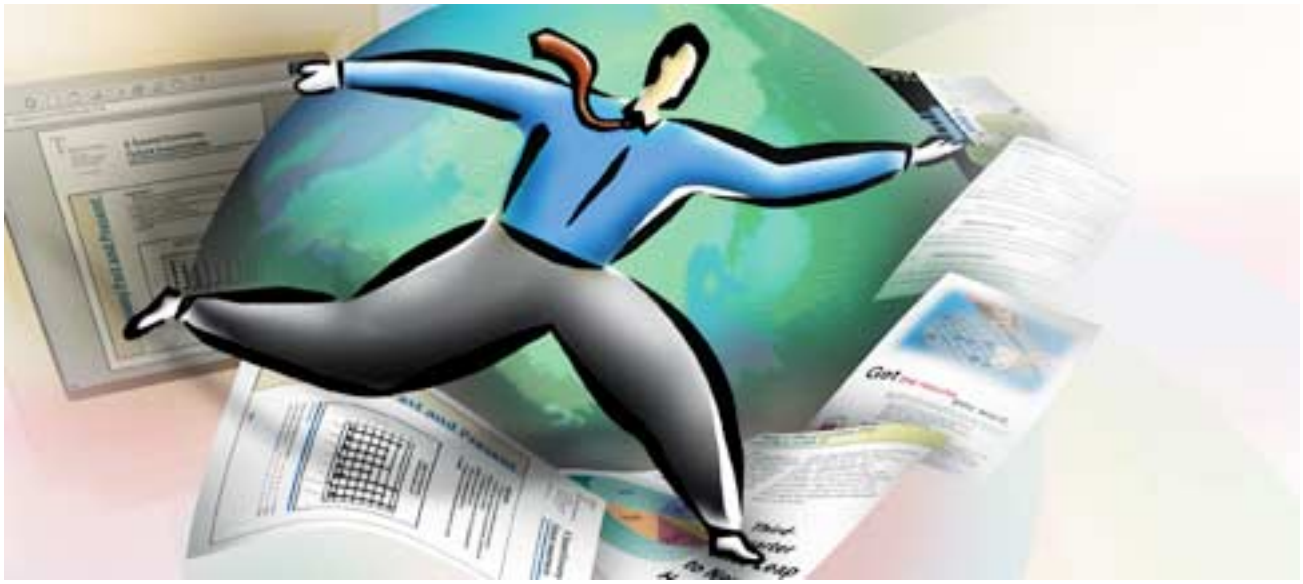
Transparency in PDF

Adobe Developer Technologies

Technical Note #5407



Revised: 30 November 2000

.

Contents

Transparency in PDF

November 30, 2000

1 Introduction

This document describes how the Adobe® imaging model is extended to include transparency and specifies how transparency is represented in the Adobe Portable Document Format (PDF). Under the transparency model, graphics objects do not necessarily obey a strict opaque painting model; instead, they can blend in interesting ways with other objects that overlap.

Support for transparency is first introduced in PDF version 1.4, which will be the native file format for Adobe Acrobat® 5. At the time of this writing, Acrobat 5 has not yet been introduced. The first Adobe application to support PDF transparency is Adobe Illustrator® 9, which supports PDF 1.4 as a native file format.

The purpose of this document is to provide preliminary information that will be useful to developers working with Illustrator 9. This document describes only the transparency features of PDF 1.4. Other features of PDF 1.4 will be published at some future time in conjunction with the introduction of Acrobat 5.

1.1 About This Document

The transparency extensions to the Adobe imaging model are very general. This document describes the general model to the extent necessary to understand the PDF extensions for transparency. However, it does not describe the realization of the transparency extensions in applications such as Adobe Illustrator or Adobe Photoshop®.

This document also does not cover how the transparency model is to be implemented. We use implementation-like descriptions at various points to describe

how things work, but this is only for the purpose of elucidating the behavior of the model. Please bear in mind that the actual implementation will almost certainly be different from what might be implied in these descriptions.

This document is organized as follows:

- Section 2, "Overview," introduces the basic concepts of the transparency model and its associated terminology, including *shape*, *opacity*, *alpha*, *blend mode*, *stack*, *backdrop*, and *group*.
- Section 3, "Color Compositing Computations," describes the mathematics of computing a result color as a function of source and backdrop colors, alphas, and blend mode.
- Section 4, "Shape and Opacity Computations," continues in the same vein and covers the related shape and opacity computations.
- Section 5, "Groups," introduces the concept of groups and covers their properties and semantics.
- Section 6, "Soft Masks," covers the creation and use of masks to specify position-dependent shape and opacity.
- Section 7, "Color Space and Color Rendering Issues," describes the interactions between the transparency model and other aspects of color specification and rendering in the existing Adobe imaging model.
- Section 8, "Overview of PDF Extensions," gives a brief overview of how the transparency extensions to the Adobe imaging model are represented in PDF.
- Section 9, "PDF Specification," gives a detailed description of the PDF extensions, organized according to *PDF Reference*.
- Section 10, "Terminology Summary," is an alphabetized summary of terminology used in this document.
- Section 11, "Compatibility," discusses compatibility with PDF 1.3 and with the PostScript® language.
- Section 12, "Overprinting, Erasing, and Transparency," presents details of the existing overprinting and erasing rules in PDF 1.3 and their equivalent representation in PDF 1.4 as a form of transparency.
- There is a revision history at the end.

1.2 Related Documents

The PDF and PostScript specifications mentioned below are available on the Adobe Solutions Network (ASN) Developer Program web site, located at:

< http://partners.adobe.com/asn/developer/ >

The official specification for PDF is now the book *PDF Reference, second edition*, published in July 2000 by Addison-Wesley (and also available on-line). That book is a self-contained reference for PDF, incorporating all information about the Adobe imaging model that formerly was documented only in *PostScript Language Reference, third edition*. Additionally, the PDF material has been extensively reorganized and revised. *PDF Reference, second edition* supersedes the former *Portable Document Format Reference Manual, version 1.3*.

This document refers to the PDF specification as, for example, *PDF Reference*, Section 1.4 [1.7]. The first number refers to a section in the new *PDF Reference, second edition*. The second number (in brackets) refers to a section in the old *Portable Document Format Reference Manual, version 1.3*, where the closest equivalent material can be found.

PDF Reference, second edition (PDF version 1.3)
Addison-Wesley, June 2000

Portable Document Format Reference Manual, version 1.3
March 11, 1999
Superseded by *PDF Reference, second edition*.

PostScript Language Reference, third edition
Addison-Wesley, February 1999

Compositing Digital Images
T. Porter & T. Duff, Computer Graphics (ACM), vol. 18 no. 3, July 1984

1.3 Intellectual Property

The information in this document is subject to the copyright permissions stated in *PDF Reference*, Section 1.4 [1.7]. Additionally, developers should be aware that many of the transparency extensions to the Adobe imaging model are the subject of patents and patents pending by Adobe Systems. The permission to use

the copyrighted material in the PDF specification does not include the right to use any Adobe patents, except as may be permitted by an official Adobe Patent Clarification Notice (published at Adobe's web site or elsewhere).

2 Overview

This section introduces the general concepts behind the transparency extensions to the Adobe imaging model. Subsequent sections go into the model in greater detail. The technical terms introduced here are summarized in Section 10, "Terminology Summary."

2.1 Basic Concepts

The existing Adobe imaging model paints objects (fills, strokes, text, images), possibly clipped by a path, opaquely onto a page. One can think of the objects on a page as forming a *stack*, where the stacking order is defined to be the order in which the objects are specified, bottommost object first. At any given point on the page, the color of the page is defined to be the color of the topmost enclosing object, disregarding any overlapping objects lower in the stack. This effect can be—and often is—realized simply by rendering objects directly to the page in the order in which they are specified.

Under the transparency imaging model, all of the objects in a stack can potentially contribute to the result. At any given point, the color of the page is defined to be the result of combining the colors of all enclosing objects in the stack, according to some *compositing* rules that the transparency model defines.

Note: *The order in which objects are specified determines the stacking order, but not necessarily the order in which objects are actually painted onto the page. In particular, the model does not require the implementation to rasterize objects immediately or to commit to a raster representation at any time prior to rendering the entire stack onto the page. This is important, since rasterization often causes significant loss of information and precision that is best avoided during intermediate stages of the transparency computation.*

A given object is composited with a *backdrop*. Ordinarily, the backdrop consists of the stack of all objects that have been specified previously; the result is then treated as the backdrop for compositing the next object. However, within certain kinds of groups (see below), a different backdrop is chosen.

When an object is composited with the backdrop, the color at each point is modified by a function called the *blend mode*, which is a function of both the object's color and the backdrop color. The blend mode determines how colors interact; different blend modes can be used to achieve a variety of useful effects. A single blend mode is in effect for compositing all of a given object, but different blend modes can be applied to different objects.

Compositing of an object with the backdrop is mediated by two scalar quantities called *shape* and *opacity*. Conceptually, for each object, these quantities are defined at every point in the plane, just as if they were additional color components. (In actual practice, they are often obtained from auxiliary sources, rather than being intrinsic to the object itself.)

Both shape and opacity vary from 0 (no contribution) to 1 (maximum contribution). At any point where either the shape or the opacity is 0, the color is undefined. At any point where the shape is 0, the opacity is also undefined. The shape and opacity themselves are subject to compositing rules, so that the stack also has a shape and opacity at each point.

An object's opacity, in combination with the backdrop's opacity, determines the relative contributions of the backdrop's color, the object's color, and the blended color to the computed composite color. An object's shape then determines the degree to which the composite color replaces the backdrop color. Shape values of 0 and 1 identify points that lie "outside" and "inside" a familiar sharp-edged object, but intermediate values are useful in defining soft-edged objects.

Shape and opacity are very similar concepts. In fact, in most situations, they can be combined into a single value, called *alpha*, which controls both the color compositing computation and the fading between the object and the backdrop. However, there are a few situations in which they must be treated separately; see Section 5.5, "Knockout Groups." Moreover, raster-based implementations must maintain a separate shape parameter in order to do anti-aliasing properly; it is therefore convenient to have it be an explicit part of the model.

One or more consecutive objects in a stack can be collected together into a *transparency group*, hereafter referred to simply as *group*. The group as a whole can have various properties that modify the compositing behavior of objects within a group and their interactions with the backdrop of the group. Additionally, an additional blend mode, shape, and opacity can be associated with the group as a

whole and used when compositing the group with its backdrop. Groups can be nested within other groups, so that the group hierarchy forms a tree structure.

Note: *The concept of transparency group is independent of the existing notions of "group" or "layer" in applications such as Adobe Illustrator. Those groupings reflect logical relationships among objects that are meaningful when editing those objects, but they are not part of the imaging model.*

The color result of compositing a group can be converted to a single-component luminosity value and treated as a *soft mask*, or just mask for short. Such a mask can then be used as an additional source of shape or opacity values during subsequent compositing operations. When the mask is used as shape, this technique is known as *soft clipping*; it is a generalization of the clipping path in the existing Adobe imaging model.

The *current page* is generalized to be a group consisting of the entire stack of objects placed on the page, composited with a backdrop that is white and fully opaque. Logically, this entire stack is then rasterized, determining the actual pixel values that are to be transmitted to the output device.

Note: *In contexts where a PDF "page" is to be treated as a piece of artwork to be placed on some other page, such as an Illustrator artboard or an encapsulated PostScript (EPS) file, we do not treat it as a page but as a group, whose backdrop may be defined differently from a page.*

2.2 Notation

The following are conventions for variable names used in this document. In general, a lowercase letter represents a scalar quantity, such as an opacity. An uppercase letter represents an *n*-tuple of scalar values, such as a color.

In the descriptions of the basic color compositing computations, color values are generally represented by the letter *C*, with a mnemonic suffix that indicates which of several color values is being referred to; for instance, *Cs* stands for "source color." Shape and opacity values are represented with the letters *f* (for "*f*orm factor") and *q* (for "opa*q*ueness"), with a mnemonic suffix, such as *qs* for "source opacity." The symbol α (alpha) stands for the product of the corresponding shape and opacity values.

In the descriptions of group transparency, the basic formulas are recast as recurrence relations and augmented with other formulas specifying group behavior. Here, variables have a numeric subscript indicating the position in the stack that the quantity is associated with, with the bottommost object numbered 0. Thus, Cs_i stands for "source color of the *i*th object in the stack."

In certain computations, one or more variables may have undefined values; for instance, when opacity is zero, the corresponding color is undefined. A value can also be undefined if it results from a division by zero. In any formula that uses the undefined value, the value has no effect on the ultimate result because it is subsequently multiplied by zero or otherwise cancelled out.

The important point is that any arbitrary value can be chosen for an undefined value, but the computation must not malfunction due to exceptions caused by overflow or division by zero. Additionally, it is convenient to adopt the convention that 0 / 0 = 0.

3 Color Compositing Computations

The primary change in the imaging model that comes with adding transparency is in how colors are painted. In the transparent model the result of painting, the *result color*, is a function of both the color being painted, the *source color*, and the color it is being painted over, the *backdrop color*. Both of these colors may vary as a function of position on the page, but for the purposes of this section we will concentrate our attention on some fixed point in the page and assume a fixed source and backdrop color.

Other parameters in this computation are the *alpha*, which specifies the relative contributions of the source and backdrop colors, and the *blend mode*, which allows one to customize how the source and backdrop colors are combined in the painting operation. This *color compositing function*, or just *compositing function* for short, determines the color result of a painting operation:

$$Cr = \left(1 - \frac{\alpha s}{\alpha r}\right) \cdot Cb + \frac{\alpha s}{\alpha r} \cdot ((1 - \alpha b) \cdot Cs + \alpha b \cdot B(Cb, Cs))$$

where the variable definitions are given in Table 1.

TABLE 1 Variables in the color compositing formula

VARIABLE	MEANING
αb	Backdrop alpha
αr	Result alpha
αs	Source alpha
$B(Cb, Cs)$	Function implementing the blend mode
Cb	Backdrop color
Cr	Result color
Cs	Source color

This is actually a simplified form of the compositing function where the shape and opacity values are combined and represented as a single alpha value. The more general form is presented later. This function is based on the Porter & Duff **over** operation, extended to include a blend mode in the region of overlapping coverage.

The following sections elaborate on the meaning and implications of these formulas.

3.1 Blending Color Space

First, note that the compositing function operates on colors. If the colors are represented by more than one scalar value then the computation treats them as vector quantities. To be precise, *Cb*, *Cr*, *Cs*, and $B(Cb, Cs)$ will all have n elements, where n is the number of components in the color space used for compositing. The above formula is then a vector function: the ith component of *Cr* is obtained by plugging in the ith components of *Cs*, *Cb* and $B(Cb, Cs)$.

Thus, the result of the computation will depend on the color space in which the colors are represented. For this reason, the color space used to represent colors for this computation is explicitly made part of the model and is called the *blending color space*. When necessary, source colors are converted to the blending color space prior to the compositing computation.

The following PDF color spaces are supported as blending color spaces: **DeviceGray**, **DeviceRGB**, **DeviceCMYK**, **CalGray**, **CalRGB**, and the equivalent **ICCBased** color spaces (including calibrated CMYK). The **Lab** space and the **ICCBased** spaces that represent lightness and chromaticity separately (such as Lab, Luv, and HSV) are not allowed, because the compositing computations in such spaces do not give meaningful results when done on a per-component basis. Additionally, blending can be done on spot colors individually, as specified in **Separation** and **DeviceN** color spaces.

The blend mode functions assume that the range per color component is from 0 to 1, and that the color space is additive. The former is true for all of the allowed blending color spaces, but the latter is not. In particular, the **DeviceCMYK**, **Separation**, and **DeviceN** spaces are subtractive. When performing blending operations in subtractive color spaces, we assume that the color component values are complemented before the blend mode function is applied and that the results of the function are then complemented before being used. By *complemented* we mean that a color component value c is replaced with $1 - c$.

This adjustment makes the effect of the blend modes numerically consistent across all color spaces. However, the actual visual effect produced by a given blend mode still depends on the color space. Blending in a device color space produces device-dependent results. Blending in a CIE-based color space produces results that are consistent across all devices. Additional details about color space issues are given in Section 7, “Color Space and Color Rendering Issues.”

3.2 Blend Mode

The $B(Cb, Cs)$ term of the compositing function is used to customize the blending operation. This function of two colors is called the *blend mode*. Abstractly, this could be any function of the source and backdrop colors that returns another color. PDF defines a standard set of named functions for the blend mode; see Table 2 and Table 3.

All of the blend modes in Table 2 are defined by a scalar function that is applied separately to each color component, expressed in additive form:

$$cr = B(cb, cs)$$

where the lowercase *cr*, *cs*, and *cb* denote one component of the colors *Cr*, *Cs*, and *Cb*. Such a blend mode is called *separable*. This is in contrast to a function

where the result for a particular component is a function of components other than the corresponding component in the backdrop and source colors. (In principle, a blend mode could have a different function for each component and yet remain separable; however, none of the blend modes listed below have that property.) A separable blend mode can be used with any color space, since it applies to any number of components. Only separable blend modes can be used when blending spot colors.

Some of the separable blend modes are defined by actual mathematical formulas; the rest are defined only by a description of their intended effects.

TABLE 2 Separable blend modes

NAME	RESULT
Normal	Normal(cb, cs) = cs Replaces the backdrop color by the source color.
Compatible	Similar to **Normal**, but it consults the overprint control parameters to produce overprinting or erasing behavior compatible with PDF 1.3. See Section 7.3, "Overprinting and Erasing," and "Blend Modes" on page 47.
Multiply	Multiply(cb, cs) = $cb \cdot cs$ Multiples the backdrop and source color values. The result color is always at least as dark as either of the two argument colors. Multiplying any color with black produces black. Multiplying any color with white leaves the color unchanged. Painting successive overlapping objects with a color other than black or white produces progressively darker colors.
Screen	Screen(cb, cs) = $cb + cs - cb \cdot cs$ = $1 - (1 - cb) \cdot (1 - cs)$ Multiplies the complements of the backdrop and source color values. The result color is always at least as light as either of the two argument colors. Screening with black leaves the color unchanged. Screening with white produces white. The effect is similar to projecting multiple photographic slides simultaneously onto a single screen.
Overlay	Multiplies or screens the colors, depending on the backdrop color. Source colors overlay the backdrop while preserving the highlights and shadows of the backdrop. The backdrop color is not replaced but is mixed with the source color to reflect the lightness or darkness of the backdrop color.

SoftLight Darkens or lightens the colors, depending on the source color value. The effect is similar to shining a diffused spotlight on the backdrop.

If the source color is lighter than 0.5, the backdrop is lightened, as if it were dodged. This is useful for adding highlights to a scene. If the source color is darker than 0.5, the backdrop is darkened, as if it were burned in. Painting with pure black or white produces a distinctly darker or lighter area but does not result in pure black or white.

HardLight Multiplies or screens the colors, depending on the source color value. The effect is similar to shining a harsh spotlight on the backdrop.

If the source color is lighter than 0.5, the backdrop is lightened, as if it were screened. This is useful for adding highlights to a scene. If the source color is darker than 0.5, the backdrop is darkened, as if it were multiplied. This is useful for adding shadows to a scene. Painting with pure black or white produces pure black or white.

ColorDodge Brightens the backdrop color to reflect the source color. Painting with black produces no change.

ColorBurn Darkens the backdrop color to reflect the source color. Painting with white produces no change.

Darken $\text{Darken}(cb, cs) = \min(cb, cs)$
Selects the darker of the backdrop and source colors. The backdrop is replaced with the source where the source is darker; otherwise it is left unchanged.

Lighten $\text{Lighten}(cb, cs) = \max(cb, cs)$
Selects the lighter of the backdrop and source colors. The backdrop is replaced with the source where the source is lighter; otherwise it is left unchanged.

Difference $\text{Difference}(cb, cs) = |cb - cs|$
Subtracts the source color from the backdrop color or the backdrop color from the source color, depending on which has the greater brightness value. Painting with white inverts the backdrop color; painting with black produces no change.

Exclusion Produces an effect similar to but lower in contrast than the Difference mode. Painting with white inverts the backdrop color; painting with black produces no change.

Table 3 lists a standard set of non-separable blend modes. Their effects are described, but no mathematical formulas are given. These modes all entail conversion to and from an intermediate hue, saturation, and luminance representation. Since the non-separable blend modes consider all color components in combination, their computation depends on the blending color space in which those colors are interpreted.

TABLE 3 Non-separable blend modes

NAME	RESULT
Hue	Creates a color with the luminance and saturation of the backdrop color and the hue of the source color.
Saturation	Creates a color with the luminance and hue of the backdrop color and the saturation of the source color. Painting with this mode in an area of the backdrop that is a pure gray (no saturation) produces no change.
Color	Creates a color with the luminance of the backdrop color and the hue and saturation of the source color. This preserves the gray levels of the backdrop and is useful for coloring monochrome images and for tinting color images.
Luminosity	Creates a color with the hue and saturation of the base color and the luminance of the blend color. This produces an inverse effect from that of the Color mode.

Note: *For illustrations of the visual effects of the blend modes, see the Adobe Photoshop User Guide.*

3.3 Interpretation of Alpha

The color compositing function (repeated below) produces a result color that is a weighted average of the source color, the backdrop color, and the $B(Cb, Cs)$ term, with the weighting controlled by the source and backdrop alphas.

$$Cr = \left(1 - \frac{\alpha s}{\alpha r}\right) \cdot Cb + \frac{\alpha s}{\alpha r} \cdot ((1 - \alpha b) \cdot Cs + \alpha b \cdot B(Cb, Cs))$$

The simplest blend mode, **Normal**, is defined by $B(Cb, Cs) = Cs$. With this blend mode, the compositing formula collapses to a simple weighted average of the source and backdrop colors, controlled by the source and backdrop alpha values.

If the blend mode is a more interesting function of the source and backdrop colors, the source and backdrop alphas control whether the effect of the blend mode is fully realized or is toned down by mixing the result with the source and backdrop colors. With any blend mode, $\alpha s = 0$ or $\alpha b = 0$ results in no blend mode effect; $\alpha s = 1$ and $\alpha b = 1$ results in maximum blend mode effect.

Mathematically, the influence of the source and backdrop colors is controlled by the source and backdrop alphas, respectively. The influence of the blend function is controlled by the product of the source and backdrop alphas.

Another variable, the result alpha, also appears in the function. This is actually a computed result, described in Section 4, “Shape and Opacity Computations.” The result color is normalized by the result alpha. This ensures that when this color and alpha are subsequently used together in another compositing operation, the color’s contribution will be correctly represented. Note that if the result alpha is zero, the result color is undefined.

The above formula is a simplification of the following one, which presents the relative contributions of backdrop, source, and blended colors in a more straightforward fashion:

$$\alpha r \cdot Cr = (1 - \alpha s) \cdot \alpha b \cdot Cb + (1 - \alpha b) \cdot \alpha s \cdot Cs + \alpha b \cdot \alpha s \cdot B(Cb, Cs)$$

The simplification requires a substitution based on the alpha compositing formula, which is presented in the next section.

4 Shape and Opacity Computations

So far, we have covered the generation of the color that results when a source color is composited with a backdrop color. This section describes the derivation of the alpha values that control the compositing process.

As indicated earlier, alpha is actually a combination of shape and opacity; it is defined simply to be their product. Thus, we define:

$$\alpha b = fb \cdot qb$$
$$\alpha r = fr \cdot qr$$
$$\alpha s = fs \cdot qs$$

We now describe the various shape and opacity values individually. Once again, keep in mind that conceptually these values are computed for every point on the page.

4.1 Source Shape and Opacity

The shape and opacity values can come from several sources. The transparency model defines three independent sources for each. However, the PDF representation imposes some limitations on the ability to specify all of these sources independently.

- *Object shape*. Elementary objects, such as strokes, fills, and text, have an intrinsic shape, whose value is 1 for points inside the object and 0 outside. Similarly, a masked image with a binary mask (as in PDF 1.3) has a shape that is 1 in the unmasked portions and 0 in the masked portions. The shape of a group object is the union of the shapes of the objects it contains.

  Note: *Mathematically, elementary objects have "hard" edges, with shape value either 0 or 1 at any given point. However, when such objects are rasterized to device pixels, the shape values along the boundaries may take on fractional values, representing fractional coverage of those pixels. When such anti-aliasing is performed, it is important to treat the fractional coverage as shape, not as opacity.*

- *Mask shape*. There can be an additional source of shape values varying by position, independent of the object itself. (How such a mask might be generated is discussed in Section 6, "Soft Masks.") Using such a mask to modify the shape of some object or group is called *soft clipping*. It can produce effects such as a gradual transition between an object and its backdrop, as in a vignette.

- *Constant shape*. This is a scalar value that simply modifies the source shape value at every point. The constant shape is just a convenience, since its effect could be simulated with a mask that has the same value everywhere.

- *Object opacity.* Elementary objects have an opacity of 1 everywhere. The opacity of a group object is the result of the opacity computations for all the objects it contains.
- *Mask opacity.* This is an additional source of opacity values varying by position, independent of the object itself.
- *Constant opacity.* This is a scalar value that simply modifies the source opacity value at every point. It is useful to think of this value as the "current opacity," analogous to the "current color," used when painting elementary objects.

The range of all of the above shape and opacity inputs is from 0 to 1, and the default value for all of them is 1. The intent is that any of the inputs described above will make the painting operation more transparent as it goes towards 0. If more than one input goes towards 0 then the result is compounded. This is achieved mathematically simply by multiplying the three inputs of each type, producing intermediate values called the *source shape* and the *source opacity.*

$$fs = fj \cdot fm \cdot fk$$
$$qs = qj \cdot qm \cdot qk$$

where the variable definitions are given in Table 4.

TABLE 4 Variables in the source shape and opacity computations

VARIABLE	MEANING
fj	Object shape
fk	Constant shape
fm	Mask shape
fs	Source shape
qj	Object opacity
qk	Constant opacity
qm	Mask opacity
qs	Source opacity

4.2 Computing the Result Shape and Opacity

In parallel with computing a result color, the painting operation also computes a *result shape* and a *result opacity* value. These values define the shape and opacity associated with the result color.

Compositing of shape and opacity values is done by the *union function*:

$$\text{Union}(b, s) = 1 - (1 - b) \cdot (1 - s) = b + s - b \cdot s$$

where b and s are the backdrop and source values to be composited. This can be thought of as an "inverted multiply": it is just a multiply with the inputs and outputs complemented. The result tends toward 1; if either input is 1 then the result is 1. This is a generalization of the conventional concept of "union" for opaque shapes.

The result shape and opacity are given by:

$$fr = \text{Union}(fb, fs)$$
$$qr = \frac{\text{Union}(fb \cdot qb, fs \cdot qs)}{fr}$$

where the variable definitions are given in Table 5.

TABLE 5 Variables in the result opacity computation

VARIABLE	MEANING
fb	Backdrop shape
fr	Result shape
fs	Source shape
qb	Backdrop opacity
qr	Result opacity
qs	Source opacity

These formulas can be interpreted as follows:

- The result shape is simply the union of the backdrop and source shapes.
- The result opacity is the union of the backdrop and source opacities, each of whose contribution is determined by its respective shape. The result is then normalized by the result shape. This ensures that when this shape and opacity are subsequently used together in another compositing operation, the opacity's contribution will be correctly represented.

Since alpha is just the product of shape and opacity, it can easily be shown that

$$\alpha r = \text{Union}(\alpha b, \alpha s)$$

This formula can be used whenever the independent shape and opacity results are not needed.

4.3 Summary of Compositing Computations

Below is a summary of all the computations from the previous section and this one. They are given in an order such that no variable is used before it is computed; also, some of the formulas have been rearranged to simplify them.

$$\text{Union}(b, s) = 1 - (1 - b) \cdot (1 - s) = b + s - b \cdot s$$
$$fs = fj \cdot fm \cdot fk$$
$$qs = qj \cdot qm \cdot qk$$
$$fr = \text{Union}(fb, fs)$$
$$\alpha b = fb \cdot qb$$
$$\alpha s = fs \cdot qs$$
$$\alpha r = \text{Union}(\alpha b, \alpha s)$$
$$qr = \frac{\alpha r}{fr}$$
$$Cr = \left(1 - \frac{\alpha s}{\alpha r}\right) \cdot Cb + \frac{\alpha s}{\alpha r} \cdot ((1 - \alpha b) \cdot Cs + \alpha b \cdot B(Cb, Cs))$$

For a list of the variables used in these formulas, see the tables in the preceding sections; the information is summarized in Table 13 on page 66.

5 Groups

A *group* is a sequence of consecutive objects in a stack that are collected together and composited to produce a single color, shape, and opacity at each point. The result is then treated as if it were a single object for subsequent compositing operations. This facilitates creating independent pieces of artwork, each composed of many objects, and then combining them, possibly with additional transparency effects during the combination. Groups can be nested within other groups; this is a strict nesting, so that the group hierarchy forms a tree structure.

The objects contained within a group are treated as a separate stack, called the *group's stack*. The objects in the stack are composited against some initial backdrop (discussed later), producing a composite color, shape, and opacity for the group as a whole. The result is an object whose shape is the union of the shapes of all constituent objects and whose color and opacity are the result of the compositing operations. This object is in turn composited with the group's backdrop in the usual way.

In addition to the computed color, shape, and opacity, the group as a whole can have several additional attributes:

- All of the input variables that affect the compositing computation for an object can also be applied when compositing the group with its backdrop. These include mask and constant shape, mask and constant opacity, and blend mode.
- The group can be *isolated* or *non-isolated*, determining the initial backdrop against which the group's stack is composited.
- The group can be *knockout* or *non-knockout*, determining whether the objects within the group's stack are composited with one another, or only with the group's backdrop.
- An isolated group can specify its own blending color space, independent of the blending color space of the group's backdrop.
- Instead of being composited onto the current page, a group's results can be used as a source of shape or opacity values for creating a *soft mask*, described in Section 6, "Soft Masks."

The next section introduces some new notation for dealing with group compositing. The following section describes the group compositing function for a non-

isolated, non-knockout group. Subsequent sections describe the special properties of groups having the isolated and knockout attributes.

5.1 Notation

Since we are now dealing with multiple objects at a time, it is useful to have some notation for distinguishing among them. Accordingly, we alter the variables used earlier to include a subscript that indicates the associated object's position in the stack. The subscript 0 indicates the initial backdrop; subscripts 1 through *n* indicate the bottommost through topmost objects in an *n*-element stack; subscript *i* indicates the object that is currently of interest. Additionally, we drop the *b* and *r* suffixes from the variables αb, *Cb*, *fb*, *qb*, αr, *Cr*, *fr* and *qr*; other variables retain their suffixes.

This convention permits the compositing formulas to be restated as recurrence relations among elements of a stack. For instance, the result of the color compositing computation for object *i* is denoted by C_i (previously *Cr*). This computation takes as one of its inputs the immediate backdrop color, which is the result of the color compositing computation for object $i-1$; this is denoted by C_{i-1} (previously *Cb*).

The revised formulas for a simple stack (not including any groups) are:

$$fs_i = fj_i \cdot fm_i \cdot fk_i$$

$$qs_i = qj_i \cdot qm_i \cdot qk_i$$

$$f_i = \text{Union}(f_{i-1}, fs_i)$$

$$\alpha s_i = fs_i \cdot qs_i$$

$$\alpha_i = \text{Union}(\alpha_{i-1}, \alpha s_i)$$

$$q_i = \frac{\alpha_i}{f_i}$$

$$C_i = \left(1 - \frac{\alpha s_i}{\alpha_i}\right) \cdot C_{i-1} + \frac{\alpha s_i}{\alpha_i} \cdot ((1-\alpha_{i-1}) \cdot Cs_i + \alpha_{i-1} \cdot B_i(C_{i-1}, Cs_i))$$

where the variable definitions are given in Table 6. Compare these with the formulas summarized in Section 4.3, "Summary of Compositing Computations."

TABLE 6 Revised variables for basic compositing computations

VARIABLE	MEANING
αs_i	Source alpha
α_i	Result alpha (after compositing object *i*)
$B_i(C_{i-1}, Cs_i)$	Function implementing the blend mode
Cs_i	Source color
C_i	Result color (after compositing object *i*)
fj_i	Object shape
fk_i	Constant shape
fm_i	Mask shape
fs_i	Source shape
f_i	Result shape (after compositing object *i*)
qj_i	Object opacity
qk_i	Constant opacity
qm_i	Mask opacity
qs_i	Source opacity
q_i	Result opacity (after compositing object *i*)

5.2 Group Structure and Nomenclature

As indicated earlier, the elements of a group are treated as a separate stack, the group's stack. Those objects are composited (against a selected initial backdrop, to be described), and the resulting color, shape, and opacity are then treated as if they belonged to a single object. The resulting object is in turn composited with the group's backdrop in the usual way.

This manipulation entails interpreting the stack as a tree. For an *n*-element group that begins at position *i* in the stack, it treats the next *n* objects as an *n*-element substack, whose elements are given an independent numbering of 1 through *n*. Those objects are removed from the object numbering in the containing stack. They are replaced by the group object, numbered *i*, followed by the remaining

objects that are painted on top of the group, renumbered starting at $i + 1$. This operation applies recursively to any nested groups. Henceforth, the term *element* (denoted E_i) refers to a member of some group; it can itself be either an object or a group.

From the perspective of a particular element in a nested group, there exist three different backdrops that are interesting to talk about:

- *Group backdrop*—the result of compositing all elements up to but not including the first element of the group. (This definition is altered if the parent group is a knockout group; see Section 5.5, "Knockout Groups.")
- *Initial backdrop*—a backdrop that is selected for compositing the group's first element. This is either the same as the group backdrop (non-isolated group) or a fully transparent backdrop (isolated group).
- *Immediate backdrop*—the result of compositing all elements of the group up to but not including the current element of interest.

When all elements of a group have been composited, the result is treated as if the group were a single object, which is then composited with the group backdrop. (Note that this operation occurs regardless of whether the group backdrop or a transparent backdrop was chosen as the initial backdrop for compositing the elements of the group. There is a special correction to ensure that the backdrop's contribution to the overall result is applied only once.)

5.3 Group Compositing Computations

The color and opacity of a group are defined by the *group compositing function*:

$$\langle C, f, \alpha \rangle = \text{Composite}(C_0, \alpha_0, G)$$

where the variable definitions are given in Table 7.

TABLE 7 Arguments and results of group compositing function

VARIABLE	MEANING
α_0	Alpha of the group's backdrop
α	Computed alpha of the group, to be used as the object alpha when the group itself is treated as an object

C_0	Color of the group's backdrop
C	Computed color of the group, to be used as the source color when the group itself is treated as an object
f	Computed shape of the group, to be used as the object shape when the group itself is treated as an object
G	The group: a compound object consisting of all the elements $E_1 \ldots E_n$ of the group—the n constituent objects' colors, shapes, opacities, and blend modes

Note that the opacity is not given explicitly as an argument or result of this function. When needed, the opacity can be computed by dividing the alpha by the associated shape. Almost all of the computations use the product of shape and opacity rather than opacity by itself, so it is usually convenient to keep track of shape and alpha, rather than shape and opacity.

The result of calling the Composite function is then treated as if it were an object, which is composited with the group's backdrop according to the normal formulas. In those formulas, the returned color C is treated as the source color Cs; the returned shape and alpha, f and α, are treated as the object shape and alpha, fj and αj.

The definition of the Composite function (for a non-isolated, non-knockout group) is as follows:

Initialization:

$$fg_0 = \alpha g_0 = 0$$

For each group element E_i in G, $i \in [1, n]$:

$$\langle Cs_i, fj_i, \alpha j_i \rangle = \begin{cases} \text{Composite}(C_{i-1}, \alpha_{i-1}, E_i) \text{ if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape} \cdot \text{opacity) of } E_i \text{ otherwise} \end{cases}$$

$$fs_i = fj_i \cdot fk_i \cdot fm_i$$

$$\alpha s_i = \alpha j_i \cdot (fk_i \cdot qk_i) \cdot (fm_i \cdot qm_i)$$

$$fg_i = \text{Union}(fg_{i-1}, fs_i)$$

$$\alpha g_i = \text{Union}(\alpha g_{i-1}, \alpha s_i)$$

$$\alpha_i = \text{Union}(\alpha_0, \alpha g_i)$$

$$C_i = \left(1 - \frac{\alpha s_i}{\alpha_i}\right) \cdot C_{i-1} + \frac{\alpha s_i}{\alpha_i} \cdot ((1 - \alpha_{i-1}) \cdot Cs_i + \alpha_{i-1} \cdot B_i(C_{i-1}, Cs_i))$$

Result:

$$C = C_n + (C_n - C_0) \cdot \left(\frac{\alpha_0}{\alpha g_n} - \alpha_0\right)$$

$$f = fg_n$$

$$\alpha = \alpha g_n$$

where the variable definitions are given in Table 8 (in addition to the ones in Table 7).

TABLE 8 Variables in the group compositing function

VARIABLE	MEANING
αs_i	Source alpha
αg_i	Group alpha: the accumulated source alphas of group elements E_1 through E_i only, excluding the initial backdrop α_0
αj_i	Object alpha for E_i,—the product of the object shape and object opacity. This is an intrinsic attribute of an elementary object (one that isn't a group); it is a computed result for a group.

α_i	Accumulated alpha (after compositing object *i*), including the initial backdrop α_0
$B_i(C_{i-1}, Cs_i)$	Function implementing the blend mode for E_i
Cs_i	Source color for E_i. This is an intrinsic attribute of an elementary object; it is a computed result for a group.
C_i	Accumulated color (after compositing object *i*), including the initial backdrop
E_i	Element *i* of the group. This is a compound variable representing the color, shape, opacity, and blend mode parameters that either are intrinsic to the object or are associated input variables.
fj_i	Object shape for E_i. This is an intrinsic attribute of an elementary object (one that isn't a group); it is a computed result for a group.
fk_i	Constant shape for E_i
fm_i	Mask shape for E_i
fs_i	Source shape
fg_i	Group shape: the accumulated source shapes of group elements E_1 through E_i only, excluding the initial backdrop
qk_i	Constant opacity for E_i
qm_i	Mask opacity for E_i

As stated above, E_i is a compound variable representing an element of a group. If the element is itself a group, it represents all the elements of that group. When Composite is called with E_i as an argument, this means to pass the entire group that E_i represents. This group is represented by the *G* variable inside the recursive call to Composite; it is expanded and its elements are denoted by $E_1 \ldots E_n$.

Note that the elements of a group are composited onto a backdrop that includes the group's initial backdrop. This is to achieve the correct effects of the blend modes, most of which are dependent on both the source and backdrop colors being blended. (This feature is what distinguishes a non-isolated group from an isolated group, discussed in the next section.)

Special attention should be directed to the formulas at the end that compute the *C*, *f*, and α results that are returned from the Composite function. Essentially, they remove the contribution of the group backdrop from the computed results.

This ensures that when the group itself is subsequently composited with that backdrop (possibly with additional shape or opacity inputs or a different blend mode), the backdrop's contribution is included only once.

For color, the backdrop removal is accomplished by an explicit calculation, whose effect is essentially the reversal of compositing with **Normal** blend mode. The formula is a simplification of the following formulas that present this operation more intuitively:

$$bf = \frac{(1 - \alpha g_n) \cdot \alpha_0}{\text{Union}(\alpha_0, \alpha g_n)}$$

$$C = \frac{C_n - bf \cdot C_0}{1 - bf}$$

where *bf* is the backdrop fraction, that is, the relative contribution of the backdrop color to the overall color.

For shape and alpha, backdrop removal is accomplished by maintaining two sets of variables to hold the accumulated values. The group shape and alpha, fg_i and αg_i, accumulate only the shape and alpha of the group elements, excluding the group backdrop; their final values become the group results returned by Composite. The complete alpha, α_i, includes the backdrop contribution as well; its value used in the color compositing computations. (There is never any need to compute the corresponding complete shape, f_i, that includes the backdrop contribution.)

As a result of these corrections, the effect of compositing objects as a group is the same as compositing them separately, without grouping, if all of the following conditions are true:

- The group is non-isolated and has the same knockout attribute as its parent group (see the next two sections).
- When compositing the group's results with the group backdrop, the Normal blend mode is used and the shape and opacity inputs are always 1.

5.4 Isolated Groups

An isolated group is one whose elements are composited onto a fully transparent initial backdrop rather than onto the group's backdrop. The resulting source col-

or, object shape, and object alpha for the group are therefore independent of the group backdrop. The only interaction with the group backdrop occurs when the group's computed color, shape, and alpha are then composited with it.

In particular, the special effects produced by the blend mode of objects within the group take into account only the intrinsic colors and opacities of those objects; they are not influenced by the group's backdrop. For example, applying the **Multiply** blend mode to an object in the group will produce a darkening effect upon other objects lower in the group's stack, but it won't produce that effect on the group's backdrop.

The effect of an isolated group can be represented by a simple object that directly specifies a color, shape, and opacity at each point. This so-called "flattening" of an isolated group is sometimes useful for import and export of fully-composited artwork in applications. Additionally, a group that specifies an explicit blending color space must be an isolated group.

For an isolated group, the group compositing function is altered simply by adding one statement to the initialization:

If the group is isolated: $\alpha_0 = 0$

That is, the initial backdrop on which the elements of the group are composited is transparent, rather than being inherited from the group's backdrop. This substitution also makes C_0 undefined, but the normal compositing formulas take care of that. Additionally, the result computation for C automatically simplifies to $C = C_n$, since there is no backdrop contribution to be factored out.

5.5 Knockout Groups

In a knockout group, each individual element is composited with the group's initial backdrop, rather than with the stack of preceding elements in the group. When objects have binary shapes (1 for "inside," 0 for "outside"), each object "knocks out" the effects of any overlapping elements within the same group. At any given point, only the topmost object enclosing the point contributes to the result color and opacity of the group as a whole.

This model is similar to the opaque painting model of the existing Adobe imaging model, except that the "topmost object wins" rule applies to both the color and the opacity. Knockout groups are useful in composing a piece of artwork

from a collection of overlapping objects, where the topmost object in any overlap completely obscures the objects underneath. At the same time, the topmost object interacts with the group's initial backdrop in the usual way, with its opacity and blend mode applied as appropriate.

The concept of "knockout" is generalized to accommodate fractional shape values. In that case, the immediate backdrop is only partially knocked out and replaced by only a fraction of the result of compositing the object with the initial backdrop.

The restated group compositing function deals with knockout groups by introducing a new variable, *b*, which is a subscript that specifies which previous result to use as the backdrop in the compositing computations: 0 in a knockout group; $i - 1$ in a non-knockout group. When $b = i - 1$, the formulas simplify to the ones given in Section 5.3, "Group Compositing Computations."

In the general case, the computation proceeds in two stages:

1. Composite the object with the group's initial backdrop, but disregarding the object's shape and using a source shape value of 1 everywhere. This produces unnormalized temporary alpha and color results, αt and Ct. (For color, this computation is essentially the same as the unsimplified color compositing formula given in Section 3.3, "Interpretation of Alpha," but using a source shape of 1.)

$$\alpha t = \text{Union}(\alpha g_b, qs_i)$$
$$Ct = (1 - qs_i) \cdot \alpha_b \cdot C_b + qs_i \cdot ((1 - \alpha_b) \cdot Cs_i + \alpha_b \cdot B_i(C_b, Cs_i))$$

2. Compute a weighted average of this result with the object's immediate backdrop, using the source shape as the weighting factor. Then normalize the result color by the result alpha.

$$\alpha g_i = (1 - fs_i) \cdot \alpha g_{i-1} + fs_i \cdot \alpha t$$
$$\alpha_i = \text{Union}(\alpha_0, \alpha g_i)$$
$$C_i = \frac{(1 - fs_i) \cdot \alpha_{i-1} \cdot C_{i-1} + fs_i \cdot Ct}{\alpha_i}$$

This averaging computation is performed for both color and alpha. The above formulas show this averaging directly. The formulas given below in Section 5.6, "Summary of Group Compositing Computations," are slightly altered to use source shape and alpha, rather than source shape and opacity, avoiding the need to compute a source opacity value explicitly. (Note that *Ct* there is slightly different from *Ct* above; it is premultiplied by fs_i.)

The extreme values of the source shape produce the straightforward "knockout" effect. That is, a shape of 1 ("inside") yields the color and opacity that result from compositing the object with the initial backdrop. A shape of 0 ("outside") leaves the previous group results unchanged. The existence of the knockout feature is the main reason for maintaining a separate shape value, rather than only a single alpha value that combines shape and opacity. The separate shape value must be computed in any group that is subsequently used as an element of a knockout group.

A knockout group can be non-isolated or isolated; that is, *isolated* and *knockout* are independent attributes of a group. A non-isolated knockout group composites its topmost enclosing element with the group's backdrop; an isolated knockout group composites the element with a transparent backdrop.

Note: *When a non-isolated group is nested within a knockout group, the initial backdrop of the inner group is the same as the initial backdrop of the outer group; it is not the immediate backdrop of the inner group. This non-obvious nesting behavior is a consequence of the formulas for Composite when b = 0.*

5.6 Summary of Group Compositing Computations

The following restatement of the group compositing function also takes isolated groups and knockout groups into account. See Table 7 on page 21 and Table 8 on page 23 for the definitions of the variables.

$\langle C, f, \alpha \rangle = \text{Composite}(C_0, \alpha_0, G)$

Initialization:

$fg_0 = \alpha g_0 = 0$

If the group is isolated: $\alpha_0 = 0$

For each group element E_i in G, $i \in [1, n]$:

$$b = \begin{cases} 0 \text{ if the group is knockout} \\ i-1 \text{ otherwise} \end{cases}$$

$$\langle Cs_i, fj_i, \alpha j_i \rangle = \begin{cases} \text{Composite}(C_b, \alpha_b, E_i) \text{ if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape} \cdot \text{opacity) of } E_i \text{ otherwise} \end{cases}$$

$$fs_i = fj_i \cdot fk_i \cdot fm_i$$

$$\alpha s_i = \alpha j_i \cdot (fk_i \cdot qk_i) \cdot (fm_i \cdot qm_i)$$

$$fg_i = \text{Union}(fg_{i-1}, fs_i)$$

$$\alpha g_i = (1 - fs_i) \cdot \alpha g_{i-1} + (fs_i - \alpha s_i) \cdot \alpha g_b + \alpha s_i$$

$$\alpha_i = \text{Union}(\alpha_0, \alpha g_i)$$

$$Ct = (fs_i - \alpha s_i) \cdot \alpha_b \cdot C_b + \alpha s_i \cdot ((1 - \alpha_b) \cdot Cs_i + \alpha_b \cdot B_i(C_b, Cs_i))$$

$$C_i = \frac{(1 - fs_i) \cdot \alpha_{i-1} \cdot C_{i-1} + Ct}{\alpha_i}$$

Result:

$$C = C_n + (C_n - C_0) \cdot \left(\frac{\alpha_0}{\alpha g_n} - \alpha_0 \right)$$

$$f = fg_n$$

$$\alpha = \alpha g_n$$

Note: *Once again, keep in mind that these formulas are in their most general form. They can be significantly simplified when some sources of shape and opacity are not present or when shape and opacity need not be maintained separately. Furthermore, in each specific type of group (isolated or not, knockout or not), some terms of these equations cancel or drop out. An efficient implementation should use the derived simplified equations.*

5.7 Page Group

All of the elements painted directly onto a page—both top-level groups and top-level objects that are not part of any group—may be treated as if they were contained in an isolated group *P*, which in turn is composited with a context-dependent backdrop. This group is called the *page group*.

Ordinarily, the page group is imposed on media, such as paper or a display screen. The backdrop is nominally white, though varying according to actual properties of the media. However, some applications may choose to provide a different backdrop, such as a checkerboard that is useful for visualizing the effects of transparency in the artwork.

Note: *A "page" of a PDF file can also be treated as a graphics object that is to be used as an element of a page of some other document. This case arises, for example, when placing a PDF file containing a piece of artwork produced by Illustrator into a page layout produced by InDesign™. In that situation, the PDF "page" is not composited with white, as described below; instead, it is treated as an ordinary transparency group, which is composited with its backdrop in the normal way.*

The color *C* of the page at a given point is defined by a simplification of the general group compositing formula:

$$\langle Cg,\ fg,\ \alpha g \rangle = \text{Composite}(U, 0, P)$$
$$C = (1 - \alpha g) \cdot W + \alpha g \cdot Cg$$

where the variable definitions are given in Table 9. The first formula computes the color and alpha for the group given a transparent backdrop—in effect, treating *P* as an isolated group. The second formula composites the results with the context-dependent backdrop (using the equivalent of **Normal** blend mode).

TABLE 9 Variables for page group computation

VARIABLE	MEANING
αg	Computed alpha of the group
C	Color of the page
Cg	Computed color of the group

fg	Computed shape of the group (this result is discarded)
P	A group consisting of all elements $E_1 \ldots E_n$ in the page's top-level stack
U	An undefined color (which is not used, since the α_0 argument of Composite is 0)
W	The initial color of the page, which is nominally white but can be context-dependent depending on properties of the media or the needs of the application.

If not otherwise specified, the page group's color space is inherited from the native color space of the output device—that is, a device color space, such as **DeviceRGB** or **DeviceCMYK**. It is often preferable to specify an explicit color space, particularly a CIE-based color space, to ensure more predictable results of the compositing computations within the page group. In this case, all page-level compositing is done in that color space, with the entire result then converted to the native color space of the output device before being composited with the context-dependent backdrop. This case also arises when the page is not actually being rendered but is converted to a "flattened" representation in an opaque imaging model, such as PostScript.

6 Soft Masks

Earlier sections have mentioned the mask shape, *fm*, and mask opacity, *qm*, that are contributors to the source shape and opacity. These enable shape and opacity to originate from a source that is independent of the objects being composited. A soft mask (or just mask for short) defines values that can vary across different points on the page. The word "soft" emphasizes that the mask value at a given point is not just 0 or 1 but can take on fractional values.

A mask used as a source of shape values is also called a soft clip. The term *soft clip* arises from analogy with the "hard" clipping path of the existing Adobe imaging model. The soft clip is a generalization of the hard clip: a hard clip can be represented as a soft clip having shape value 1 inside and 0 outside the clipping path. Everywhere inside a hard clipping path, a source object's color replaces the backdrop; everywhere outside, the backdrop shows through unchanged. With a soft clip, on the other hand, one can create a gradual transition between an object and the backdrop, such as in a vignette. (Although "soft clip" suggests a vignette

around an enclosed area, in fact the value of the mask can be an arbitrary function of position.)

A mask is typically the only means of providing position-dependent opacity, since elementary objects do not have intrinsic opacity.

A mask can be defined by creating a group and then painting objects into it, thereby defining color, shape, and opacity in the usual way. The resulting group can then be treated as a mask by following one of the two procedures described below.

6.1 Mask from group alpha

The color, shape, and opacity of the group *G* are first computed by the usual formula:

$$\langle C,\ f,\ \alpha \rangle\ =\ \text{Composite}(C_0,\ \alpha_0,\ G)$$

where C_0 and α_0 are an arbitrary backdrop whose value does not contribute to the eventual result. The *C*, *f*, and α results are the group's color, shape, and alpha, with the backdrop factored out.

The mask value at each point is then derived from the alpha of the group. In this case, the group's color is not used, so there is no need to compute it. The alpha value is passed through a separately-specified transfer function, enabling the masking effect to be customized.

6.2 Mask from group luminosity

The group is composited with a fully-opaque backdrop of some selected color. The mask value at any given point is then defined to be the luminosity of the resulting color. This enables the mask to be derived from the shape and color of an arbitrary piece of artwork, drawn with ordinary painting operators.

The color *C* used to create the mask from a group *G* is defined by:

$$\langle Cg,\ fg,\ \alpha g \rangle\ =\ \text{Composite}(C_0,\ 1,\ G)$$

$$C\ =\ (1 - \alpha g) \cdot C_0 + \alpha g \cdot Cg$$

where C_0 is the selected backdrop color.

G can be any kind of group—isolated or not, knockout or not—with various effects in the *C* result produced in each case. The color *C* is then converted to luminosity in one of the following ways, depending on the group's color space:

- For CIE-based spaces, convert to CIE XYZ and use the *Y* component as luminosity. This procedure produces a colorimetrically correct luminosity. In the case of a PDF **CalRGB** space, the formula is:

  $$Y = Y_A \cdot A^{G_R} + Y_B \cdot B^{G_G} + Y_C \cdot C^{G_B}$$

  using components of the **Gamma** and **Matrix** entries of the color space dictionary (see *PDF Reference*, Table 4.14 [7.28]). An analogous computation applies to other CIE-based color spaces.

- For device color spaces, convert the color to **DeviceGray** by device-dependent means and use the resulting gray value as luminosity, with no compensation for gamma or other color calibration. This method makes no pretense of colorimetric correctness; it merely provides a numerically simple means to produce contone mask values. Here are some recommended formulas for converting from **DeviceRGB** and **DeviceCMYK**, respectively:

  $$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$
  $$Y = 0.3 \cdot (1 - C) \cdot (1 - K) + 0.59 \cdot (1 - M) \cdot (1 - K) + 0.11 \cdot (1 - Y) \cdot (1 - K)$$

Following this conversion, the result is then passed through a separately-specified transfer function, enabling the masking effect to be customized.

The backdrop color that is most likely to be useful is black. If the backdrop is black, any areas outside the group's shape will end up with a zero luminosity value in the resulting mask. If we view the contents of the group as a positive mask, this result matches our expectations with respect to the points that are outside the shape.

7 Color Space and Color Rendering Issues

This section describes the interactions between the transparency model and other aspects of color specification and rendering in the Adobe imaging model.

7.1 Color Spaces

A group can have either an explicitly declared color space or the color space inherited from the parent group. In either case:

1. A source object's color is converted to the group's color space if necessary.
2. The blending and compositing computations are done in that color space (see Section 3.1, "Blending Color Space").
3. The group's resulting color is interpreted as being in that color space when the group is subsequently composited with the backdrop.

Under this arrangement, we envision that all or most of a piece of artwork will be created in a single color space—most likely, the working color space of the application that generated it. Use of multiple color spaces typically will arise only when assembling independently-produced artwork onto a page. When the complete artwork is placed on a page, the conversion from the group's color space to the page's device color space will be done as the last step, without any further transparency compositing.

The transparency model does not impose any requirement that such a convention be followed. The reason for adopting it is to avoid loss of color information and introduction of errors due to performing unnecessary conversions.

Only isolated groups may have an explicitly declared color space; non-isolated groups must inherit their color space from the parent group. Use of an explicit color space in a non-isolated group would require converting colors from the backdrop's color space to the group's color space in order to perform the compositing computations. This may not be possible, since some color conversions are one-way.

The choice of group color space will have significant effects on the results that are produced. In particular:

- As indicated in Section 3.1, "Blending Color Space," if the group uses a device color space, then the transparency compositing computations will produce device-dependent results.
- In order for the compositing computations to work in a device-independent way, the group's color space must be a CIE-based color space.

- A consequence of choosing a CIE-based group color space is that only CIE-based color spaces can be used to specify the colors of objects within the group. This is because conversion from device to CIE-based colors is not possible in general; the defined conversions work only in the opposite direction.
- The compositing computations and blend modes generally compute linear combinations of color component values, based on the assumption that the color component values themselves are linear. Therefore, it is usually best to choose a group color space that is linear (that is, it has a linear gamma curve). If a non-linear color space is chosen, the results are still well-defined, but the appearance may not match the user's expectations.

Note: *In this connection, note that RGB is a non-linear CIE-based color space, hence possibly unsuitable for use as a group color space.*

Note: *An implementation of the transparency model is advised to use as much precision as possible to represent colors during the compositing computations and in the accumulated group results. More precision is needed for intermediate results than is typically used to represent either the original source data or the final rasterized results. This is to minimize accumulation of roundoff errors and to avoid additional errors that arise from the use of linear group color spaces.*

7.2 Spot Colors

The preceding discussion about color spaces has been concerned about *process colors*—that is, the colors that are produced by combinations of the device's process colorants. (Process colors are sometimes called "composite colors," but we will avoid that term here due to possible confusion with the transparency compositing operations.) Process colors may be specified directly in the device's color space (such as **DeviceCMYK**), or they may be produced by conversion from some other color space, such as a CIE-based color space (**CalRGB** or **ICCBased**). Whatever means is used to specify them, process colors are subject to conversion to and from the group's color space.

A *spot color* is an additional color component, independent of the color components that are used to produce process colors. A spot color can represent either an additional separation that is to be produced or an additional colorant that is to be applied to the composite page. The color component value for a spot color is called a *tint*, representing the concentration of the spot colorant. (Tints are conventionally represented as subtractive values, not additive.)

Spot colors are inherently device-dependent and are not always available. In the Adobe imaging model as represented in PostScript and PDF, each use of a spot color (in a **Separation** or **DeviceN** color space) is accompanied by an *alternate color space* and a function for mapping tint values into that color space. This enables the spot color to be approximated with process colors when the spot colorant is not available in the device.

Spot colors can be accommodated straightforwardly in the transparency model, except for issues relating to overprinting, discussed in Section 7.3, "Overprinting and Erasing." If an object that is an element of a group is painted with a spot color, one of two things can happen:

- The group maintains a separate color value for each spot color component, independent of the group's color space. Effectively, the spot color passes directly through the group hierarchy to the device, with no color conversions performed; however, it is still subject to blending and compositing with other objects that use the same spot color.
- The spot color is converted to its alternate color space. The resulting color is then subject to the usual compositing rules for process colors.

Only a single shape value and opacity value are maintained at each point in the computed group results; they apply to both process and spot color components. In effect, every object is considered to paint every color component that exists. Where no value has been specified for a given color component in a given object, an additive value of 1 (or subtractive tint value of 0) is assumed.

For instance, when painting an object with a color specified as **DeviceCMYK** or **ICCBased**, the process color components are painted as specified and the spot color components are painted with additive value 1. Likewise, when painting an object with a color specified as **Separation**, the named spot color is painted as specified and the other components (both process and other spot colors) are painted with additive value 1. The consequences of this are discussed in Section 7.3, "Overprinting and Erasing."

The existing Adobe imaging model also allows the process color components to be addressed individually, as if they were spot colors. For instance, one can specify a **Separation** color space named Cyan, which paints just the cyan component in a CMYK output device.

This is very difficult to extend to work in transparency groups. In general, the color components in a group are not the process colorants themselves, but are converted to process colorants only after completion of all color compositing computations for the group (and perhaps some of the group's parents as well). For instance, if the group's color space is **ICCBased**, the group has no Cyan component to be painted.

Therefore, treating a process color component as if it were a spot color is permitted only within a group that inherits the native color space of the output device. Attempting to do so in a group that specifies its own color space will result in conversion of the requested spot color to its alternate color space.

7.3 Overprinting and Erasing

This section addresses the relationship between overprinting and transparency and discusses how the Adobe imaging model is altered to deal with it. This discussion is limited to what can be represented in PDF. There is an entirely separate question of how applications can use transparency to simulate the effects of overprinting colorants; that topic is beyond the scope of this specification.

Background

PDF has an *overprint* parameter and an *nonzero overprint mode* parameter in the graphics state. They are documented in *PDF Reference*, Section 4.5.6. (The older PDF 1.3 specification lacks detailed documentation of these parameters. The overprint parameter, but not the nonzero overprint mode, also exists in PostScript; it is documented in *PostScript Language Reference, third edition*, Section 4.8.5.)

Briefly, in the existing imaging model, painting an object causes some specific set of device colorants to be marked according to the current color space and color value. The remaining colorants are either erased or left unchanged, according to whether the overprint parameter is *false* or *true*. The nonzero overprint mode parameter additionally enables this selective marking of colorants to be applied to individual components of **DeviceCMYK** according to whether the component value is zero or nonzero.

This model of overprinting is very device-dependent. It deals directly with the painting of device colorants, independent of the color space in which source col-

ors have been specified. It primarily addresses production needs, not design intent. Overprinting is usually reserved for an opaque colorant or for a very dark color such as black. It is also invoked during late-stage production operations, such as trapping, when the actual set of device colorants has already been determined.

It is best to think of transparency as taking place in appearance space, but overprinting of device colorants in device space. This means that colorant overprint decisions should be made at output time based on the actual resultant colorants of any transparency compositing operation. On the other hand, effects similar to overprinting can be achieved in a device-independent manner by taking advantage of blend modes; this is described below.

Use of Blend Modes to Erase or Overprint

As indicated in Section 7.2, “Spot Colors,” each object paints every color component that exists—both the process color components in the group’s color space and any available spot color components. For color components whose value has not been specified, a source color of 1 is assumed. When objects are fully opaque and **Normal** blend mode is used, this has the effect of erasing those components. This is consistent with the existing opaque imaging model when the overprint flag is turned off.

The transparency model defines some blend modes, such as **Darken**, that can be used to achieve effects similar to overprinting. The definition of **Darken** is:

$$\text{Darken}(cb, cs) = \min(cb, cs)$$

If the blend mode is **Darken**, its result will always be the same as the backdrop color when the source color is 1, as it is for all unspecified color components. When the backdrop is fully opaque, painting with a source color of 1 and the **Darken** blend mode leaves the result color unchanged from the backdrop. This is consistent with the existing opaque imaging model when the overprint flag is turned on.

If the object or backdrop are not fully opaque, the above actions are altered correspondingly. That is, the “erasing” effect is reduced, and “overprinting” an object with color value 1 may affect the result color. While these results may or may not be useful, they lie outside the realm of the erasing and overprinting that are defined in the existing opaque imaging model.

When process colors are erased or overprinted (because a spot color is being painted), the blending computations described above are done componentwise in the group's color space. If the group's color space is different from the native color space of the output device, its components are not the actual process colorants of the output device; the blending computations affect the process colorants only after the group's results are converted to the device color space. Thus, the effect is different from erasing or overprinting the device's process colorants directly. On the other hand, this is a fully general operation that works uniformly, regardless of the type of object and regardless of what computations produced the source color.

The above discussion has concentrated on the color components whose values have *not* been specified and that are to be either erased or left unchanged. The **Normal** or **Darken** blend modes used for those purposes may not be suitable for use on the components whose color values *have* been specified. In particular, the **Darken** blend mode for those components would preclude overpainting a dark color with a lighter color. Moreover, some other blend mode may be specifically desired for those components.

PDF provides means to specify only one blend mode, which always applies to process colorants and sometimes applies to spot colorants as well. Specifically, only *white-preserving* blend modes can be used for spot colors—that is, functions having the property that $B(1, 1) = 1$. If a non-white-preserving blend mode is specified, it applies only to the process color components; **Normal** blend mode is substituted for the spot colors. This ensures that when objects accumulate in an isolated group, the accumulated values for unspecified components remain 1. The group's results can then be overprinted using **Darken** (or other useful modes) while avoiding unwanted interactions with the components whose values were never specified within the group.

Interpretation of Overprint Parameters

The previous section describes how effects similar to overprinting can be achieved using blend modes. Those methods do not make direct use of the overprint flag and nonzero overprint mode; they are usable only by transparency-aware applications.

PDF provides for compatibility with PDF 1.3 overprint control by defining a special **Compatible** blend mode that consults the overprint control parameters to

compute its result. See “Blend Modes” on page 47 for the precise definition of the **Compatible** blend mode.

7.4 Rendering Parameters

PDF has several graphics state parameters dealing with the rendering of color: halftone, transfer functions, color rendering intent, undercolor removal, and black generation. How should these parameters work in the presence of transparency?

The problem is this: The rendering parameters can be specified on a per-object basis; they control how that object will be rendered. When all objects are opaque, it is easy to define what this means. When they are transparent, more than one object can contribute to the color at a given point. It is unclear which rendering parameters to apply in an area where transparent objects overlap. (Devising a way to “blend” the effects of these parameters seems hopelessly difficult.)

Furthermore, the operations that the rendering parameters control—halftoning in particular—can be performed only when the final color at a given point is known. When objects are transparent, rendering of an object does not occur at the time the object is specified, but at some later time. The implementation must keep track of the rendering parameters at each point from the time they are specified until the time the rendering actually occurs. In other words, rendering parameters must be associated with regions of the page rather than with individual objects.

At the same time, we would like the transparency imaging model to be compatible with the existing opaque imaging model in the case that only opaque objects are painted.

For the halftone and transfer function parameters, the problem is solved in the following way. Conceptually, there is a map over the entire page defining the rendering parameters to be used at each point. This map is defined as follows:

- Initially, the rendering parameters for the entire page have default values, which are the values in effect at the beginning of the current page.
- If the topmost object at a given point is fully opaque, then the rendering parameters associated with the object are used at that point. This provides exact compatibility with the existing opaque imaging model. The definitions of “topmost object” and “fully opaque” are given below.

Only elementary objects define the rendering parameters map; the rendering parameters associated with a group object are ignored. At a given point, the *topmost object* is the topmost elementary object in the entire page stack that has a nonzero object shape value (*fj*) at that point (in other words, the point is "inside" the object). An object is considered to be *fully opaque* if its source alpha is 1, its blend mode is **Normal** or **Compatible**, and each direct ancestor group likewise specifies a source alpha of 1 and a blend mode of **Normal** or **Compatible**. These conditions ensure that only the object itself contributes to the color at that point; it completely obscures the backdrop.

The color rendering intent parameter needs to be handled a little differently. This parameter determines how to convert from CIE-based color spaces to device color spaces. It is needed at the moment such a conversion must be performed—that is, when painting an elementary object with a CIE-based color or a group whose color space is CIE-based into a parent group or page having a device color space. This is unlike the halftone and transfer function parameters, whose values are used when rasterization is performed.

The color rendering intent to be used during a conversion is determined by defining a map that resembles the one used for the halftone and transfer function but is applied only at the group level:

- When painting an elementary object (not a group) having a CIE-based color directly into a group or page having a device color space, the color rendering intent associated with that object is used directly. This occurs without regard for whether the object is fully opaque.
- When painting a group object whose color space is CIE-based into a group or page having a device color space, the color rendering intent to use at each point is determined by the topmost object within the group (including nested subgroups) if it is fully opaque. If there is no such object, the color rendering intent to use is the one associated with the group object itself.

A similar approach works for the undercolor removal and black generation functions, which are applied only during conversion from **DeviceRGB** to **DeviceCMYK** color spaces.

8 Overview of PDF Extensions

The preceding sections have described the transparency model at a fairly abstract level, with relatively little mention of how it is represented in PDF. This section introduces the PDF representation; it is organized according to the presentation of the transparency model above. The next section specifies the PDF extensions for transparency in detail; it is organized according to the presentation in *PDF Reference*.

8.1 Color Compositing Computations

The object color *Cs* comes from the usual sources of color, that is, the current color in the graphics state or the source samples in an image. The backdrop color *Cb* is the result of previous painting operations.

The blending color space is an attribute of the transparency group within which an object is painted. The page as a whole is also treated as a group (the page group) that has a color space attribute. If not otherwise specified, the page group's color space is inherited from the native color space of the output device.

The blend mode *B*(*Cb*, *Cs*) is a parameter in the graphics state. It is a name selecting among a fixed enumeration of blend modes.

8.2 Shape and Opacity Computations

Every object painted by PDF painting operators has a shape value *fj* at each point, defined as follows:

- The shape of an object defined by a path (fill, stroke, text) and painted with a simple paint is always 1 inside and 0 outside the path.
- The shape of an image is nominally 1 inside and 0 outside the image rectangle. This can optionally be reduced by a binary mask accompanying the image, which is specified by either an explicit mask or a color key (using the PDF 1.3 masked image feature).
- The shape of an image mask is 1 for painted regions and 0 for masked regions.
- The shape of a shading object (painted by the **sh** operator) is 1 inside and 0 outside the bounds of the shading's painting geometry, disregarding the **Background** entry.

- The shape of an object painted with a tiling pattern is recursively determined by the objects that define the pattern.
- The shape of an object painted with a shading pattern is the intersection of the object itself and the geometry of the shading.

All elementary objects have an intrinsic opacity *qj* of 1 everywhere. Any desired opacity less than 1 must be applied by means of the *qk* and *qm* parameters described below.

The graphics state contains a constant alpha value and a position-dependent alpha mask. There is a separate parameter that specifies whether these alpha sources are to be treated as shape (*fk* and *fm*) or as opacity (*qk* and *qm*).

The constant alpha is specified by two simple scalar parameters in the graphics state: one for strokes; one for all nonstroking operations. It is reasonable to think of these parameters as the "current alpha," analogous to the current color used when painting elementary objects. (Note, however, that the nonstroking alpha is also applied when painting a group's results onto the backdrop.)

There can be at most one mask input in any compositing operation. It can be specified in either of the following ways:

- The graphics state contains a soft mask parameter. If present, its value is a *soft-mask dictionary*, which includes a transparency group that is to be used as a position-dependent source of alpha values. By this means, an arbitrary collection of objects can be used to define the shape or opacity that is then imposed on the objects painted onto the page.
- An image XObject can contain a *soft-mask image*, specified as a subsidiary image XObject. This mask, if present, overrides the PDF 1.3 masked image feature. Either form of mask in the image XObject overrides the soft mask parameter in the graphics state.

Note: *The limitation of one mask per compositing operation causes no loss of generality in the transparency model. A soft mask can also be applied to a group, and groups can be nested to multiple levels.*

8.3 Groups

A group is represented in PDF as a *transparency-group XObject*—a form XObject having some additional transparency-related attributes, which are carried in a separate *group attributes* subdictionary. The elements of the group consist of the graphics objects that are painted by execution of the XObject's content stream. The results of the group compositing computations—color, shape, and opacity—are then painted into the group's parent group or page. The **Do** operator invokes both of the above operations.

The entries in the group attributes dictionary include:

- Color space for blending and compositing, if different from the one used in the parent group or page
- Isolated and knockout boolean attributes

For a non-isolated group, the group's backdrop is defined as the computed color, shape, and opacity of everything that has been painted into the parent group or page prior to execution of the **Do** operator.

8.4 Soft Masks

A soft mask is represented by a *soft-mask dictionary.* This dictionary contains the following:

- A transparency-group XObject describing the group that is to be used as the source of position-dependent mask values
- Backdrop color and blending color space to use for the group compositing operation
- Other entries controlling conversion from the group results to mask values

A soft mask created in this way can be established as the soft mask parameter in the graphics state. This causes it to be treated as a position-dependent source of shape or opacity (*fm* or *qm*) in subsequent compositing operations.

8.5 Color Space and Color Rendering Issues

The issues and proposed model for color spaces, spot colors, and overprinting are quite thoroughly covered in Section 7, “Color Space and Color Rendering Issues.” The PDF representation follows straightforwardly from that description.

8.6 Limitations in the PDF Transparency Model

The PDF realization of the general transparency model has several limitations in order to simplify the representation and the implementation. The following is a summary of these limitations, which are further described elsewhere in this specification.

- In any given transparency compositing operation, there is at most one mask input, which can be treated as either shape or opacity. (This is in addition to the shape and opacity that are intrinsic to the source object itself.)
- There are not independent parameters for constant shape and constant opacity. Instead, there is a constant alpha value, which can be treated as either shape or opacity (the other parameter is implicitly 1).
- There is only one blend mode specified per object, regardless of the number of color components that are affected when the object is painted. It always applies to the process color components. If it is a separable, white-preserving blend mode, then it also applies to any spot color components that exist; otherwise, **Normal** blend mode is substituted for those components. There is a fixed enumeration of blend modes, with no provision for specifying an arbitrary function as a blend mode.

9 PDF Specification

This section describes the additions and changes to the PDF specification. It is organized according to the presentation in *PDF Reference, second edition*. The numbers in brackets refer to sections in the older *Portable Document Format Reference Manual, version 1.3*, where the closest equivalent material can be found.

9.1 Adobe Imaging Model—2.1.2

[This section needs to be revised to incorporate the transparency extensions to the Adobe imaging model, described here in Section 2, “Overview.”]

9.2 Page Tree (Page Objects)—3.6.2 [6.4]

[Add the following entry to Table 3.17:]

Table 3.17 [6.5] Entries in a page dictionary

KEY	TYPE	VALUE
Group	dictionary	*(Optional)* A *group attributes dictionary*, which specifies the group attributes of the page group. See Table 11 on page 58 for the contents of this dictionary. See also Section 5.7, "Page Group."

9.3 Graphics Objects—4.1 [8.1]

[Replace the sentence "Each graphics object is painted...opaque painting model..." with the following:]

A PDF content stream represents a sequence of *graphics objects*. Each of these objects has a *shape*. Within the shape, every point has a *color* and an *opacity.* For some objects, the color and opacity are constant within the entire shape; for others, they can vary at different points within the shape.

In the case that all the objects are fully opaque, one can consider the objects simply to be painted onto the current page, obscuring any existing marks they may overlay. However, in the general case where transparency can occur, it is necessary to think of the objects as forming a *stack*, where the stacking order is defined to be the order in which the objects are specified, bottommost object first. All of the objects in a stack can potentially contribute to the result, according to the color, shape, and opacity compositing rules specified in Section 3, "Color Compositing Computations," and Section 4, "Shape and Opacity Computations."

[Append the following:]

These graphics objects are also treated as the elementary objects for transparency compositing purposes (subject to special treatment for text objects, described in Section 9.15). That is, all of a given object is considered to be one element of a stack. Portions of an object are not composited with one another, even if they are described in a way that would seem to cause overlaps, such as a self-intersecting path, combined fill and stroke of a path, or a shading pattern containing an overlap or fold-over.

The result of compositing a transparency-group XObject is itself treated as if it were an elementary graphics object, having shape, opacity, and color at each point. It is then composited with its parent group or page. Note that this treatment applies only to form XObjects having a **Group** entry specifying a group attributes dictionary whose **S** (Subtype) is **Transparency**. Painting an ordinary form XObject (lacking a **Group** entry) is equivalent to painting the form's constituent graphics objects individually; there is no transparency grouping behavior.

9.4 Details of Graphics State Parameters—4.3.2 [8.4]

[Add the following subsection:]

Blend Modes

The *blend mode* graphics state parameter is specified as the value of the **BM** entry in a graphics state parameter dictionary. Its value can be one of the following:

- The name of a standard blend mode, which is one of: **Compatible**, **Normal**, **Multiply**, **Screen**, **Difference**, **Darken**, **Lighten**, **ColorDodge**, **ColorBurn**, **Exclusion**, **HardLight**, **Overlay**, **SoftLight**, **Luminosity**, **Hue**, **Saturation**, and **Color**. These blend modes are explained in Section 3.2, "Blend Mode," except for the blend mode named **Compatible**, which is explained below.
- An array of one or more blend modes, each specified by a name as described above. This specifies alternate blend modes that can be used; the viewer will use the first blend mode that it recognizes in the array. If the viewer does not recognize any of the blend modes, it will use **Normal** mode. This allows a PDF file to use a new blend mode that has been introduced, while specifying reasonable fall-back behavior in a viewer that doesn't recognize the new mode.

There is only one blend mode parameter in the graphics state. This blend mode always applies to process color components and sometimes applies to spot color components as well. Specifically:

- If the blend mode is separable and white-preserving, then it applies to all spot color components that exist, as well as to process color components. A *separable* blend mode is one that is performed componentwise, with no interaction between components. A *white-preserving* blend mode is a function $B(cb, cs)$ having the property that $B(1, 1) = 1$. Of the named blend modes listed above,

the following are separable and white-preserving: **Compatible**, **Normal**, **Multiply**, **Screen**, **Darken**, **Lighten**, **ColorDodge**, **ColorBurn**, **HardLight**, **Overlay**, **SoftLight**.

- Otherwise, the blend mode applies only to process color components. Any spot color components are blended using **Normal** mode.

As explained in Section 7.2, "Spot Colors," components whose values are not explicitly specified in the current color space are implicitly painted with additive value 1 (that is, subtractive tint 0). When objects accumulate in an isolated group, the accumulated values for unspecified components remain 1 so long as only white-preserving blend modes are used. This enables the group's results to be overprinted using **Darken** (or other useful modes) while avoiding unwanted interactions with the components whose values were never specified within the group.

There is a special blend mode named **Compatible**, which is the default value of the blend mode parameter. This blend mode implements overprinting and erasing behavior that is compatible with the existing Adobe imaging model and PDF 1.3 when painting opaque objects. To do this, it consults the current values of the overprint flag (set by **op** or **OP** in a graphics state parameter dictionary) and nonzero overprint mode (set by **OPM**), as follows:

- If the object being painted is a group (not an elementary object), the **Compatible** blend mode is equivalent to **Normal**—it just returns the source color. In other words, the special compatibility behavior applies only when painting elementary objects: fills, strokes, text, images, and shading objects.
- If the overprint flag is *false*, the **Compatible** blend mode is equivalent to **Normal**.
- If the overprint flag is *true*, the overprint mode is 1 (nonzero overprint mode enabled), and the current color space and group color space are both **DeviceCMYK**, then the **Compatible** blend mode returns the backdrop color for any **DeviceCMYK** component whose (subtractive) color value is zero; it returns the source color otherwise. For spot color components, **Compatible** returns the backdrop color.
- Otherwise, the **Compatible** blend mode returns the source color for a color components that is specified in the current color space; it returns the backdrop color otherwise. For instance, if the current color space is **DeviceCMYK** or **CalRGB**, **Compatible** returns the source color for process color components

and the backdrop color for spot color components. On the other hand, if the current color space is a **Separation** color space, **Compatible** returns the source color for that spot color component and the backdrop color for process colors and all other spot color components.

In the above description, "current color space" refers to the color space used for a painting operation. This may be the color space parameter in the graphics state, a color space used implicitly by operators such as **rg**, or an attribute of an image XObject. In the case of an **Indexed** space, this refers to the underlying color space; likewise for **Separation** and **DeviceN** spaces that revert to their alternate color space.

See Section 12, "Overprinting, Erasing, and Transparency," for tables giving details of the overprinting and erasing behavior described above.

9.5 Graphics State Parameter Dictionaries—4.3.4 [7.15]

[Add the following entries to table 4.8:]

Table 4.8 [7.49] Entries in a graphics state parameter dictionary

KEY	TYPE	VALUE
ca	number	*(Optional)* Constant alpha, used in the computation of the source shape or opacity for each object painted by operations other than stroke. It must be in the range 0 to 1. Default value: 1. This parameter is implicitly reset to its default value at the beginning of execution of a transparency-group XObject.
CA	number	*(Optional)* Similar to **ca**, but applies only when stroking paths and glyph outlines.
SMask	dictionary or name	*(Optional)* Soft mask, used to provide the mask shape or opacity value at each point when computing the source shape or opacity for an object being painted. This is described by a *soft-mask dictionary*; see "Soft-Mask Dictionaries" on page 61. If this parameter is the name **None**, the mask value is implicitly 1 everywhere. Default value: **None**. This parameter is implicitly reset to its default value at the beginning of execution of a transparency-group XObject. When painting an image XObject, the soft mask parameter in the graphics state is ignored (treated the same as **None**) if the image XObject contains a **Mask** or **SMask** entry.

AIS	boolean	*(Optional)* Alpha is shape, indicating whether the sources of alpha are to be treated as shape (*true*) or opacity (*false*). This determines the interpretation of the constant alpha (**ca** and **CA**) and soft mask (**SMask**) parameters of the graphics state, as well as a soft-mask image (**SMask** entry) of an image XObject. Default value: *false*.
BM	name or array	*(Optional)* The blend mode for color compositing for each object painted. It must be the name of a blend mode or an array of alternate blend modes. See "Blend Modes" on page 47. Default value: **Compatible**. This parameter is implicitly reset to its default value at the beginning of execution of a transparency-group XObject.
TK	boolean	*(Optional)* Text knockout parameter, which determines the behavior of overlapping glyphs within a text object. If the value of **TK** is *false*, each glyph in a text object is treated as a separate elementary object; when glyphs overlap, they will composite with one another. If the value of **TK** is *true*, all the glyphs in a text object are treated together as a single elementary object; when glyphs overlap, later glyphs will knock out earlier ones in the area of overlap. See Section 9.15. Default value: *true*.

Note: *When the* **gs** *operator alters any of the above parameters, the new values completely replace the old ones. In particular, the soft mask is not intersected with its former value, as might be inferred from its role as a "soft clip" that is a generalization of the clipping path.*

Note: *Although the soft mask is defined as a parameter in the graphics state, its intended use is to clip only a single object at a time (either an elementary object or a group). If a mask is applied when painting two or more objects that overlap, the effect of the mask will multiply with itself in the area of overlap (except in a knockout group), producing a result shape or opacity that is probably not what is intended. To apply a soft mask to multiple objects, it is usually best to treat those objects as a group and apply the mask when painting the group. The foregoing considerations also apply to the constant alpha parameter in the graphics state.*

9.6 Path-Painting Operators—4.4.2 [8.6.2]

[Change the following entries in Table 4.10:]

Table 4.10 Path-painting operators

OPERANDS	OPERATOR	DESCRIPTION
—	**B**	Fill and then stroke the path, using the nonzero winding number rule to determine the region to fill. This produces the same result as specifying two path objects with the same path, painting the first with **f** and the second with **S**. Note that the filling and stroking portions of the operation consult different values of several graphics state parameters, such as color. For transparency compositing purposes, the combined fill and stroke are treated as a single graphics object. That is, the stroke replaces the fill in the area of overlap. The effect is equivalent to performing the fill and stroke as separate operations in a knockout group. (Otherwise, a non-opaque stroke would composite with the results of the fill, producing a "double border" effect. If that is the effect desired, it can be achieved by specifying the fill and stroke with separate path objects.) ***Note:*** *These transparency semantics also apply to the* **B***, **b**, *and* **b*** *operators. However, this does not change their descriptions, which are based on the behavior of the* **B** *operator.*

9.7 Clipping Path Operators—4.4.3 [8.3.1.1]

[Append the following to the first paragraph:]

In the context of the transparency model, the clipping path constrains an object's shape. The effective shape is the intersection of the object's intrinsic shape and the current clipping path: the source shape value is 0 outside this intersection. In the same vein, the shape of a transparency group is defined as the union of the shapes of all constituent objects. This shape is likewise influenced by the clipping path in effect when each of those objects is painted; it is additionally constrained by the clipping path in effect at the time of the **Do** operator, when the group's results are painted onto the backdrop.

Note: *The clipping path and the soft mask are independent parameters of the graphics state. Even though the "soft" shape mask could be considered a generalization of the "hard" clip, attempting to unify these parameters would disrupt the existing PDF graphics model unacceptably.*

9.8 Device Color Spaces—4.5.3 [7.12]

[Append the following:]

Use of device color spaces is subject to special treatment within a transparency group whose group color space is CIE-based; see “Transparency-Group XObjects” on page 57. In particular, the device color space operators should be used only if the device color spaces have been remapped to CIE-based color space by means of the **DefaultGray**, **DefaultRGB**, and **DefaultCMYK** color space resources. Otherwise, the results will be implementation-dependent and unpredictable.

9.9 CIE-Based Color Spaces (Rendering intents)—4.5.4 [8.5.1.5]

[Append the following:]

For an introduction to the problem of determining rendering parameters in the face of transparency, see Section 7.4, “Rendering Parameters.” The rendering intent parameter is needed whenever a CIE-based color must be converted to a device color space. This can occur:

- when painting an elementary object with a CIE-based color directly into a transparency group having a device color space. In this case, the rendering intent used is the current rendering intent in the graphics state at the time of the painting operation.
- when painting a group whose color space is CIE-based into a parent group having a device color space. In this case, the rendering intent used at any given point is determined by the rendering intent for the topmost enclosing object within the group (including nested subgroups), but only if that object is fully opaque. If there is no such object, the rendering intent used is the current rendering intent in effect at the time of the **Do** operator for the group.

What it means for an object to be fully opaque is explained in Section 9.18.

9.10 Special Color Spaces (Separation Color Spaces)—4.5.5 [7.12.8]

[Append the following:]

A **Separation** color space is ordinarily used to produce a spot color—an additional color component, independent of the ones that are used to produce process colors. When an object is painted transparently with a spot color, that color is composited with the corresponding spot color component of the backdrop, independent of the compositing that is performed for process colors. A spot color retains its own identity; it is not subject to conversion to or from the color space of the enclosing transparency group or page.

Any object, including a transparency group, is considered to have been painted with all color components that exist, both process and spot. Components that haven't been explicitly painted have an additive color value of 1 (that is, a subtractive tint value of 0). This has consequences that are discussed in Section 7.3, "Overprinting and Erasing."

Instead of producing a spot color, a **Separation** color space can be used to paint a single process colorant, such as the Cyan component in a device whose native color space is **DeviceCMYK**. However, this is permitted only in a group that inherits the native color space of the output device. If such a **Separation** color space is used in a transparency group that specifies its own color space, the alternate color space will be substituted.

Spot colors are never available in a transparency-group XObject that is used to define a soft mask. The alternate color space will always be substituted in that situation.

(DeviceN Color Spaces)

[Append the following:]

The transparency considerations described above for **Separation** color spaces also apply to **DeviceN** color spaces.

9.11 Overprint Control—4.5.6 [8.4.9]

[Append the following:]

The overprint control parameters (set by the **op**, **OP**, and **OPM** entries in a graphics state parameter dictionary) are respecified in terms of transparency blend

modes; see Section 7.3, “Overprinting and Erasing” and “Blend Modes” on page 47. These parameters have no effect if the blend mode is not **Compatible**.

9.12 Tiling Patterns (PatternType 1)—4.6.2 [7.17.2]

[Include the following subsection somewhere:]

Tiling Patterns and Transparency

A tiling pattern is defined by an arbitrary sequence of graphics objects that are used to paint the pattern cell, which is then replicated to tile the region being painted. Those objects can include transparent objects and transparency groups. Transparent compositing can occur both within the pattern cell and between it and the backdrop wherever the pattern is painted.

In the general case, the effect of tiling with a pattern must be as if the definition of the pattern were re-executed for each tile. This is unlike painting with a completely opaque pattern, where the pattern cell can be evaluated once and then replicated.

Note: *This can be significantly optimized in the common case in which the pattern consists entirely of objects painted with* **Normal** *blend mode. In that case, the same effect can be produced by treating the pattern cell as if it were an isolated group. The pattern cell can have color, shape, and opacity at each point, but those results do not depend on the backdrop or on the graphics state parameters at the time of use. This means that the pattern cell can be evaluated once and then replicated, just as it can for a pattern defined with opaque painting.*

The above discussion applies to both colored (**PaintType** 1) and uncolored (**PaintType** 2) tiling patterns. These two types of patterns differ in how colors are specified. A colored tiling pattern specifies colors as part of the definition of the pattern cell; an uncolored tiling pattern uses a single color that is specified separately when the pattern is used. An uncolored pattern’s definition may not specify colors; this restriction extends to any transparency group that the definition includes. However, there are no corresponding restrictions on specifying transparency parameters in the graphics state.

9.13 Image Dictionaries—4.8.4 [7.13.1]

[Add the following entry to Table 4.35:]

Table 4.35 [7.33] Entries in an image dictionary

KEY	TYPE	VALUE
SMask	stream	*(Optional)* A subsidiary image XObject defining a *soft-mask image* to be used as a source of shape or opacity values for transparency purposes. Unlike a general soft mask that can be derived from arbitrary objects (see Section 6, "Soft Masks"), this mask is defined solely by an image, whose samples are interpreted as mask values directly. The contents of the soft-mask image are described in Table 10. If this entry is absent, no soft mask is associated with the image (though the soft mask in the graphics state may still apply). If **SMask** is present, the **Mask** entry, if present, is disregarded. Furthermore, if either **SMask** or **Mask** is present, it overrides the soft mask parameter in the graphics state. (However, the other transparency-related graphics state parameters—constant alpha and blend mode—remain in effect.) The alpha is shape (**AIS**) parameter in the graphics state determines whether this soft-mask image is to be treated as shape or opacity.

Soft-Mask Images

Table 10 documents the entries in a soft-mask image dictionary. Except as noted, the syntax and semantics of the entries are the same as in a regular image XObject, documented in Table 4.35. The image XObject entries that are not mentioned here are not relevant to this type of image and are ignored.

TABLE 10 Entries in a soft-mask image dictionary

KEY	TYPE	VALUE
Type	name	*(Optional)* If present, must be **XObject**.
Subtype	name	*(Required)* Must be **Image**.
Width	integer	*(Required)* If a **Matte** entry is specified, the value of **Width** must be the same as the **Width** of the parent image. Otherwise, the **Width** of the mask image is independent of the **Width** of the parent image. Both images are mapped to the unit square in user space (as all images are), whether or not the samples coincide individually.

Height	integer	*(Required)* Same considerations as for **Width**.
BitsPerComponent	integer	*(Required)*
ColorSpace	name	*(Required)* Must be **DeviceGray**.
Decode	array	*(Optional)* Default value: [0 1].
Interpolate	boolean	*(Optional)*
ImageMask	boolean	*(Optional)* Must be *false* or absent.
Mask	stream	*(Optional)* Must be absent.
SMask	stream	*(Optional)* Must be absent.
Matte	array	*(Optional)* Specifies a color, the *matte color*, with which the image data in the parent image has been pre-blended (see below). The array must contain *n* numbers, where *n* is the number of components in the **ColorSpace** entry for the parent image; the numbers must be valid color components in that color space. If the **Matte** entry is absent, the image data is not pre-blended.

Pre-Blended Image Data

When an image is accompanied by a soft-mask image, it is sometimes advantageous for the image data to be pre-blended with some background color, called the *matte color*. Each image sample represents a weighted average of the original source color and the matte color, using the corresponding mask sample as the weighting factor. (This is a generalization of a technique that is commonly called "pre-multiplied alpha.")

If the image data is pre-blended, the matte color must be specified by a **Matte** entry in the soft-mask image XObject. The pre-blending computation, performed componentwise, is as follows:

$$c' = m + \alpha \cdot (c - m)$$

where c is the original image component value, m is the matte color component value, and α is the corresponding mask sample. The result, c', is the value that should be provided in the image source data.

Note: *This computation uses actual color component values, with the effects of the* **Filter** *and* **Decode** *transformations already performed. The computation is the same whether the color space is additive or subtractive.*

When pre-blended image data is used in transparency blending and compositing computations, the results are the same as if the original, unblended image data was used and no matte color was specified. In particular, the inputs to the blend mode function are the original color values. This may sometimes require the implementation to invert the above formula to derive c from c'. If this produces a c value that lies outside the bounds of color component values for the image color space, the results are unpredictable.

The pre-blending computation is done in the color space specified by the parent image's **ColorSpace** entry. This is independent of the group color space into which the image may be painted; if a color conversion is required, inversion of the pre-blending must precede the color conversion. If the image color space is an **Indexed** color space, the color values in the color table (not the index values themselves) must be pre-blended.

9.14 Form XObjects—4.9 [7.13.7]

[Add the following entry to Table 4.41:]

Table 4.41 [7.37] Entries in a type 1 form dictionary

KEY	TYPE	VALUE
Group	dictionary	*(Optional)* A *group attributes dictionary*, which indicates that the entire contents of the form XObject are to be treated as a group and specifies the attributes of that group. See Table 11 for the contents of this dictionary. ***Note:*** *The following paragraph refers to a PDF 1.4 feature, referenced PDF, whose specification has not yet been published.* If the form XObject contains a **Ref** entry specifying a referenced PDF file, these group attributes also apply to the referenced PDF page when it replaces the proxy. This allows an arbitrary external PDF page to be treated as a transparency group, without requiring that the referenced PDF file be modified.

Transparency-Group XObjects

A form XObject containing a **Group** entry is to be treated as a group. The value of this entry is a *group attributes dictionary* containing the entries described in Table 11. A *transparency-group XObject* is a form XObject whose group at-

tributes dictionary has subtype **Transparency**, which is the only group subtype defined at present.

A page object may also have a **Group** entry, which specifies the group attributes dictionary for the page as a whole. Some entries in the dictionary are interpreted slightly differently for a page group than for a transparency-group XObject, as indicated in the descriptions of those entries below. See also Section 5.7, "Page Group."

TABLE 11 Entries in a group attributes dictionary

KEY	TYPE	VALUE
Type	name	*(Optional)* If present, must be **Group**.
S	name	*(Required)* Group subtype. At present, the only defined value is **Transparency**; the remaining entries described below apply to this subtype. Future extensions may introduce other group subtypes, which will most likely have a different set of additional entries.
CS	name or array	*(Sometimes required, as discussed below)* Group color space. This specifies the color space into which colors are converted when painted into this group, and it defines the blending color space; see Section 3.1, "Blending Color Space." It also defines the color space of the group as a whole when it in turn is painted as an object onto the group's backdrop. **CS** may be any device or CIE-based color space that treats its components as independent additive or subtractive values in the range 0 to 1. This excludes **Lab** and lightness-chromaticity **ICCBased** color spaces (see Section 3.1, "Blending Color Space"), as well as the special color spaces **Indexed**, **Separation**, **DeviceN**, and **Pattern**. Device color spaces are subject to remapping according to the **DefaultGray**, **DefaultRGB**, and **DefaultCMYK** entries in the **ColorSpace** dictionary in the current **Resources** dictionary; see *PDF Reference*, Section 4.5.4 [7.12.12]. Ordinarily, the **CS** entry is allowed only if **I** (Isolated) is *true*, and even then it is optional. However, **CS** is required in the group attributes dictionary for a transparency-group XObject that has no parent group or page from which to inherit. This situation arises for a transparency-group XObject that is the value of the **G** entry in a soft-mask dictionary of subtype **Luminosity**; see "Soft-Mask Dictionaries" on page 61. Additionally, it is always permissible to specify **CS** in the group attributes dictionary associated with a page object, even if **I** is false or absent. In the normal case in which the page is imposed directly on the output media, the

page group is effectively isolated, regardless of the **I** value; thus, the **CS** value can take effect. But if the page is in turn used as an element of some other page and if the group is non-isolated, **CS** is ignored; the color space is inherited from the actual backdrop with which the page is composited. See Section 5.7, "Page Group."

Default value: color space of the parent group or page into which this transparency-group XObject is painted by **Do**. (The parent's color space in turn can be either explicitly specified or inherited.) If a transparency-group XObject is used as an annotation appearance, its default **CS** value inherits from the page.

I	boolean	*(Optional)* Specifies the isolated attribute of the group. This determines whether the initial backdrop for compositing the objects within the group is the group's backdrop (*false*) or a transparent backdrop (*true*). See Section 5.4, "Isolated Groups." Default value: *false*. In the group attributes dictionary for a page, the interpretation of the **I** value is slightly altered. In the normal case in which the page is imposed directly on the output media, the page group is effectively isolated, regardless of the **I** value. But if the page is in turn used as an element of some other page, the page is treated as if it were a transparency-group XObject; the **I** value is interpreted in the normal way to determine that group's isolated attribute.
K	boolean	*(Optional)* Specifies the knockout attribute of the group. This determines whether each object is composited with earlier objects in the group (*false*) or with the group's initial backdrop (*true*). See Section 5.5, "Knockout Groups." Default value: *false*.

When a transparency-group XObject is invoked by **Do**, the following actions occur in addition to the normal actions for invoking a form XObject:

1. If the value of **I** (Isolated) is *false*, the initial backdrop (within the bounding box specified by the XObject's **BBox** entry) is defined to be the accumulated color and alpha of the parent group or page—that is, the result of everything that has been painted in the parent up to that point. (However, if the parent is a knockout group, the initial backdrop is the same as the parent's initial backdrop.) If the value of **I** is *true*, the initial backdrop is defined to be transparent.
2. In the graphics state, the current nonstroking and stroking alpha are initialized to 1, the current soft mask is initialized to **None**, and the blend mode is initialized to **Compatible**.

Note: *The purpose of initializing the soft mask, alpha, and blend mode parameters in the graphics state at the beginning of execution is to ensure that these parameters aren't applied twice: once when objects are painted into the group and again when the group is painted into the parent group or page.*

3. Objects painted by operators in the transparency-group XObject's content stream are composited into the group according to the rules described in Section 3, "Color Compositing Computations," and Section 4, "Shape and Opacity Computations." The **K** (Knockout) entry in the group attributes dictionary and the transparency parameters of the graphics state contribute to this computation.
4. If **CS** (Color Space) is specified in the group attributes dictionary, all painting operators convert source colors to the specified color space prior to compositing objects into the group. The resulting color at each point in the group is interpreted in that color space. If **CS** is not specified, the prevailing color space is dynamically inherited from the parent group or page. (If not otherwise specified, the page group's color space is the device color space of the output device.)
5. After execution of the transparency-group XObject's content stream, the graphics state reverts to its former state prior to the execution of **Do** (as occurs for any form XObject, not just a transparency-group XObject). The group's shape—the union of all objects painted into the group, clipped by the XObject's bounding box—is then painted into the parent group or page, using the group's accumulated color and opacity at each point within the shape.

Note: *If a given transparency-group XObject is invoked by* **Do** *more than once, each invocation is treated as a separate transparency group. That is, the result must be as if the group were independently composited with the backdrop on each invocation. If the implementation performs any caching of rendered form XObjects, it must take this requirement into account.*

The above actions occur only for a transparency-group XObject—one having a group attributes subdictionary whose **S** (Subtype) is **Transparency**. A regular form XObject—one having no **Group** entry—is not subject to any grouping behavior for transparency purposes. That is, the graphics objects that it contains are composited individually, just as if they were painted directly into the parent group or page.

If the group's color space (either specified by **CS** or inherited) is a CIE-based color space (**CalGray**, **CalRGB**, or **ICCBased**), any use of device color spaces

for painting objects is subject to special treatment. Device colors cannot be painted directly into such a group, since there is no generally-defined method for converting them to the CIE-based color space. This problem arises in the following cases:

- **DeviceGray**, **DeviceRGB**, and **DeviceCMYK** color spaces, unless remapped to CIE-based color spaces by means of the corresponding **DefaultGray**, **DefaultRGB**, or **DefaultCMYK** color space resources.
- Operators that specify a device color space implicitly, such as **rg**, unless that space is remapped.
- Special color spaces whose underlying space is a device color space, unless that space is remapped.

We recommend that the color space remapping mechanism always be employed when defining a transparency group whose color space is CIE-based. However, if a device color is specified and is not remapped, it will be converted to the CIE-based color space in an implementation-defined fashion, producing unpredictable results.

Soft-Mask Dictionaries

Instead of being painted on the current page, a transparency-group XObject can be used as a soft mask—a source of position-dependent shape or opacity values to apply as a mask when painting other objects; see Section 6, “Soft Masks.” The soft mask is a parameter of the graphics state; see Section 9.5. Its value is a *soft-mask dictionary*, one of whose entries is the transparency-group XObject that is to be treated as the source of the mask values. The soft-mask dictionary describes how the mask is to be derived from the results of compositing the transparency group; see Table 12.

TABLE 12 Entries in a soft-mask dictionary

KEY	TYPE	VALUE
Type	name	*(Optional)* If present, must be **Mask**.
S	name	*(Required)* Subtype that specifies how the mask is to be computed. **Alpha** means to just use the group’s computed alpha (product of shape and opacity), disregarding its color. **Luminosity** means to convert the group’s computed color to a single-component luminosity value. See Section 6, “Soft Masks.”

G	stream	*(Required)* Transparency-group XObject that is to be used as the source of alpha or color values for generating the mask. If the value of **S** (Subtype) is **Luminosity**, the group attributes dictionary (value of **Group** in the transparency-group XObject) must contain a **CS** entry that defines the color space in which the compositing computation is to be done.
BC	array	*(Optional; consulted only if* **S** *(Subtype) is Luminosity)* The color that is to be used as the backdrop with which the transparency-group XObject **G** is to be composited. The value of **BC** is an array of *n* numbers, where *n* is the number of components in the color space specified by the **CS** entry in the group attributes dictionary. Default value: the color space's initial value, which represents black.
TR	function or name	*(Optional)* Transfer function, used during the conversion to mask values. This is either a function object (dictionary or stream) or the name **Identity**. The input to this function is the computed alpha or luminosity (depending on the value of the **S** (Subtype) entry), in the range 0 to 1. The output is the mask value in the range 0 to 1; it is clipped to that range if necessary. Default value: **Identity**.

If **S** (Subtype) is **Alpha**, the transparency-group XObject **G** is evaluated to compute a group alpha only; the colors of the constituent objects are ignored and the color compositing computations are not performed. The computed group alpha is then passed through the **TR** function to produce the mask. Outside the bounding box of the transparency-group XObject, the mask value is the result of presenting 0 as input to the **TR** function.

If **S** is **Luminosity**, the transparency-group XObject **G** is composited with a fully-opaque backdrop whose color is **BC** everywhere. The computed result color is then converted to a single-component luminosity value, which is passed through the **TR** function to produce the mask. Outside the bounding box of the transparency-group XObject, the mask value is the **BC** color transformed to luminosity and passed through the **TR** function.

The coordinate system of the soft mask is defined as the **Matrix** of the transparency-group XObject concatenated with user space at the moment the soft-mask dictionary is established in the graphics state by the **gs** operator.

In a transparency-group XObject that defines a soft mask, spot color components are never available, regardless of whether they are available in the group or page on which the soft mask is used. If the content stream specifies a **Separation** or

DeviceN color space that uses spot color components, the alternate color space will be substituted.

9.15 Text State Parameters and Operators—5.2 [8.7]

[Add the following somewhere:]

Text Knockout

The text knockout parameter determines what are considered to be the elementary objects for transparency compositing purposes. If text knockout is *false*, each glyph in a text object is treated as a separate elementary object; when glyphs overlap, they will composite with one another. If text knockout is *true*, all the glyphs in a text object are treated together as a single elementary object; when glyphs overlap, later glyphs will knock out earlier ones in the area of overlap. (The latter behavior is equivalent to treating the entire text object as if it were a knockout group.)

The text knockout parameter is specified as the value of the **TK** entry in a graphics state parameter dictionary. (Unlike other text state parameters, there is no special operator for specifying this parameter.) The text knockout parameter applies to entire text objects; it may not be set between the **BT** and **ET** operators that delimit a text object.

9.16 Text Rendering Mode—5.2.5 [8.7.1.7]

[Add the following somewhere:]

Rendering modes 2 and 6 specify both fill and stroke. If glyphs in the text object overlap, the result must be as if the fill and stroke are done on a glyph-by-glyph basis. That is, perform the fill and stroke for the first glyph, then the fill and stroke for the second glyph, and so on; not all of the fills followed by all of the strokes for multiple glyphs together. This produces the correct visual appearance of stacking of opaque glyphs.

For transparency compositing purposes, text rendering modes 2 and 6 have the same effect as applying the **B** operator to the glyph outline path. That is, the stroke replaces the fill in the area of overlap; it does not composite with the result

of the fill. Note that this behavior is independent of the value of the text knockout (**TK**) graphics state parameter.

9.17 Text-Showing Operators—5.3.2 [8.7.5]

[Add the following somewhere:]

Note: *Within a text object, there is no significance in the grouping of glyphs into strings presented to text-showing operators such as* **Tj**. *Showing multiple glyphs with one invocation of* **Tj** *produces the same results as showing them with a separate invocation for each glyph.*

9.18 Conversion from DeviceRGB to DeviceCMYK—6.2.3 [8.4.10]

[Append the following:]

For an introduction to the problem of determining rendering parameters in the face of transparency, see Section 7.4, “Rendering Parameters.” The undercolor removal and black generation functions are needed whenever a **DeviceRGB** color must be converted to a **DeviceCMYK** color. This can occur:

- when painting an elementary object with a **DeviceRGB** color directly into a transparency group whose color space is **DeviceCMYK**. In this case, the functions used are the current undercolor removal and black generation parameters in the graphics state at the time of the painting operation.
- when painting a group whose color space is **DeviceRGB** into a parent group whose color space is **DeviceCMYK**. In this case, the functions used at any given point are determined by the parameters in effect when painting the topmost enclosing object within the group (including nested subgroups), but only if that object is fully opaque. If there is no such object, the functions used are the ones in effect at the time of the **Do** operator for the group.

An object is considered to be fully opaque if all of the following conditions are true at the time the object is painted:

- The constant alpha parameter (**ca** or **CA**, whichever applies to the painting operation) is 1.
- The soft mask parameter (**SMask**) in the graphics state is **None**. If the object is an image, there is no **SMask** entry in the image dictionary.
- The blend mode parameter (**BM**) is either **Compatible** or **Normal**.
- The foregoing three conditions were also true at the time the group containing the object (or any direct ancestor group) was invoked by **Do**.
- If the current color is a tiling pattern, all of the objects comprising its definition also satisfy the above conditions.

9.19 Transfer Functions—6.3 [8.4.12]

[Append the following:]

For an introduction to the problem of determining rendering parameters in the face of transparency, see Section 7.4, “Rendering Parameters.” The transfer function to be used at any given point on the page is the transfer function parameter that is in effect at the time of painting the last (topmost) graphics object enclosing that point, but only if that object is fully opaque. What it means for an object to be fully opaque is explained in Section 9.18 above and further qualified below.

The transfer function to use at a given point must be determined on a per-component basis if any graphics object is painted using the **Compatible** blend mode. An object is considered opaque for a given component only if the **Compatible** blend mode returns the source color (not the backdrop color) for that component. See “Blend Modes” on page 47.

For portions of the page whose topmost object is not fully opaque or that are never painted at all, the default transfer function for the page is used.

9.20 Halftones—6.4 [8.4.13]

[Append the following:]

For an introduction to the problem of determining rendering parameters in the face of transparency, see Section 7.4, “Rendering Parameters.” The halftone to be used at any given point on the page is the halftone parameter that is in effect at the time of painting the last (topmost) graphics object enclosing that point, but only if that object is fully opaque. The conditions for applying it are the same as for the transfer function parameter, described in Section 9.19 above.

9.21 Annotation Appearances—7.4.4 [6.6.3]

[Add the following somewhere:]

An annotation appearance can include transparency if the form XObject that defines the appearance has a **Group** entry whose value is a group attributes dictionary of subtype **Transparency**, indicating that the XObject is a transparency-group XObject. This group is composited with a backdrop consisting of the page content and any other annotation appearances that are below this one. The final compositing operation is performed using a constant alpha of 1, **None** soft mask, and **Normal** blend mode.

If a transparent annotation appearance is placed on top of an annotation that is visible but is drawn without using an annotation appearance dictionary, the effect is implementation-dependent. This is because such annotations are sometimes drawn by means that do not conform to the Adobe imaging model. Also, the effect of highlighting a transparent annotation appearance is implementation-dependent.

10 Terminology Summary

Table 13 lists terms used in Section 2 through Section 7 to explain the transparency model. Where a term refers to some variable, the table gives the variable name that is conventionally used. This table does not include specific terms for PDF objects used to represent the transparency model.

TABLE 13 Terminology summary

TERM	VARIABLE(S)	MEANING
Alpha	α	The product of shape and opacity.
Backdrop		The stack of preceding objects onto which a new object is being painted.

Backdrop alpha	αb, α_{i-1}	The computed alpha of the backdrop at a given point.
Backdrop color	Cb, C_{i-1}	The computed color of the backdrop at a given point.
Backdrop fraction	bf	The fraction of the group's accumulated color that is attributable to the initial backdrop color.
Backdrop opacity	qb, q_{i-1}	The computed opacity of the backdrop at a given point.
Backdrop shape	fb, f_{i-1}	The computed shape of the backdrop at a given point.
Blend mode	$B(Cb, Cs)$	A function in the color compositing computation that customizes how source and backdrop colors combine.
Blending color space		The color space in which all colors in the compositing function are represented. Input colors are converted to this color space when necessary. Same as group color space.
Color compositing function		The function that computes the color result of painting an object over a backdrop.
Constant opacity	qk, qk_i	A simple scalar contribution to the source opacity.
Constant shape	fk, fk_i	A simple scalar contribution to the source shape.
Current page		The result produced by compositing the entire stack of objects onto a backdrop that is initially white and fully opaque.
Element	E_i	A compound variable representing all the parameters of an object being treated as an element of a group. The parameters include color, shape, and opacity (either intrinsic or computed), as well as the separately-specified shape, opacity, and blend mode parameters that are to be used when compositing the object. The element can correspond to either an elementary object or a group.
Elementary object		A unitary object in the imaging model, such as a fill, stroke, glyph, or sampled image. (Contrast with group.)
Group		A sequence of consecutive objects in a stack that are composited together and then treated as a single object to be composited with the group's backdrop.
Group alpha	αg_i	Accumulated alpha of objects within a group, excluding the group's backdrop.
Group backdrop		Ordinarily, the result of compositing all elements up to but not including the first element of the current group. However, if the parent of the current group is a knockout group, then the group backdrop is the same as the parent's group backdrop.

Group color space		A color space associated with the group as a whole. Source colors are converted into this color space if necessary. All blending and compositing computations among objects in the group occur using colors in this color space. The group's result color is interpreted as being in this color space.
Group opacity	qg_i	Accumulated opacity of objects within a group, excluding the group's backdrop. (This is rarely represented alone; it is usually combined with shape in the corresponding alpha value.)
Group shape	fg_i	Accumulated shape of objects within a group, excluding the group's backdrop.
Immediate backdrop		The result of compositing all elements of a group up to but not including the current element of interest.
Initial backdrop		A backdrop that is selected for compositing the current group's first element. This is either the same as the group backdrop (non-isolated group) or a fully transparent backdrop (isolated group).
Isolated group		A group whose elements are composited with a transparent initial backdrop, rather than with the group's backdrop.
Knockout group		A group whose elements are composited with the group's initial backdrop, rather than with the results of compositing elements lower on the group's stack.
Mask		A source of position-dependent shape or opacity values that is independent of the objects being composited.
Mask opacity	qm, qm_i	The value of an opacity mask at a given point.
Mask shape	fm, fm_i	The value of a shape mask at a given point.
Non-separable blend mode		A blend mode in which a component of the result color is a function of components other than the corresponding component of the backdrop and source colors.
Object		Either an elementary object or a group.
Object color		The intrinsic color of the source object at a given point, in its original color space.
Object opacity	qj, qj_i	The intrinsic opacity of the source object at a given point.
Object shape	qj, qj_i	The intrinsic shape of the source object at a given point.
Opacity	q	A weighting factor that determines the relative contribution of the associated color to the overall result of a color compositing computation. Its value is in the range 0 (fully transparent) to 1 (fully opaque).

Opacity compositing function		The function that computes the opacity result of painting an object over a backdrop.
Opacity mask		See mask.
Page group		A group consisting of all the objects (both elementary objects and groups) that are painted directly onto the current page.
Process color		Any color that is produced by combinations of the device's process colorants. (Contrast with spot color.)
Result alpha	αr, α_i	The result of the alpha compositing function at a given point, which becomes the backdrop alpha for the next object painted.
Result color	Cr, C_i	The result of the color compositing function at a given point, which becomes the backdrop color for the next object painted.
Result opacity	qr, q_i	The result of the opacity compositing function at a given point, which becomes the backdrop opacity for the next object painted. (This is rarely represented alone; it is usually combined with shape in the corresponding alpha value.)
Result shape	fr, f_i	The result of the shape compositing function at a given point, which becomes the backdrop shape for the next object painted.
Separable blend mode		A blend mode in which each component of the result color is a function of the corresponding component of the backdrop and source colors.
Shape	f	A weighting factor that determines the degree to which the results of a compositing operation (color and opacity) replace the backdrop. The extreme values 0 and 1 represent the familiar notions of "outside" and "inside" a hard-edge shape.
Shape compositing function		The function that computes the shape result of painting an object over a backdrop.
Shape mask		See mask.
Soft clip		See shape mask.
Soft mask		Same as mask; the "soft" adjective emphasizes that this is a continuous-tone value, not just 0 or 1.
Source alpha	αs, αs_i	The product of source shape and source opacity.
Source color	Cs, Cs_i	The intrinsic color of the source object at a given point, converted to the blending color space if necessary.
Source opacity	qs, qs_i	The product of all input opacity values: object opacity, constant opacity, and mask opacity.

Source shape	fs, fs_i	The product of all input shape values: object opacity, constant opacity, and mask opacity.
Spot color		A color that is produced by application of a separate colorant, such as special ink, that is independent of the normal process colorants.
Stack		An ordered sequence of objects painted onto the page or collected into a group.
Transparency group		See group.
Union	Union(b, s)	The function used to composite shape and opacity values. It is a generalization of the conventional concept of "union" for solid shapes.
White-preserving blend mode		A separable function $B(cb, cs)$ having the property that $B(1, 1) = 1$ for all color components (expressed in additive form).

11 Compatibility

11.1 Backward Compatibility

A PDF 1.3 or earlier viewer will ignore all transparency-related parameters, such as alpha, soft mask, and blend mode. All graphics objects, including ones defined in transparency-group XObjects, will be painted opaquely.

11.2 Forward Compatibility

The PDF 1.3 overprint control parameters (set by the **op**, **OP**, and **OPM** entries in a graphics state parameter dictionary) are respecified to work as special transparency blend modes. Their behavior should be compatible when only opaque painting operations are performed. When they are combined with other transparency operations, their behavior, though well-defined, is not a logical extension of the selective colorant marking semantics of PDF 1.3.

A process colorant (e.g., Cyan in a **DeviceCMYK** device) may not be treated as a spot color (via a **Separation** or **DeviceN** color space) within a transparency group whose color space is not inherited from the device. The alternate color space will be used instead. This isn't really a compatibility issue, since PDF 1.3 doesn't have transparency groups to begin with. The concern is about legacy artwork that might be imported and then have transparency applied to it globally.

If a transparency group's color space is CIE-based, any objects that are painted using device color spaces will be converted into the group's color space by implementation-defined means, possibly producing unexpected results. This problem can be circumvented by properly defining the **DefaultGray**, **DefaultRGB**, and **DefaultCMYK** color space resources. Once again, this is not really a compatibility issue, but it may arise when attempting to apply transparency to legacy artwork.

11.3 PostScript Printing

The PostScript language does not support the transparency model. Therefore, a PDF 1.4 viewer must have means to produce a completely opaque description of the appearance of a document that uses transparency. A similar operation can produce a PDF file that can be correctly viewed by a PDF 1.3 viewer.

This involves some combination of shape decomposition and pre-rendering to "flatten" the stack of transparent objects on a page, perform all the transparency computations, and describe the final appearance using opaque objects only. This is an irreversible operation, since all of the information about how the transparency effects were produced has been lost.

Note: *Determining if a page contains transparency needing to be "flattened" can be accomplished by straightforward analysis of the page's resources; it isn't necessary to analyze the content stream.*

In order to perform the transparency computations properly, the PDF viewer needs to know the native color space of the output device. When the viewer controls the output device directly, this is no problem. However, when the viewer is generating PostScript output, it has no way to know the native color space of the PostScript output device. Making an incorrect assumption will ruin the calibration of any CIE-based colors appearing on the page. This problem can be addressed in a couple of ways:

- If the entire page consists of CIE-based colors, then flatten the colors to a single CIE-based color space, rather than to a device color space. The preferred

color space for this purpose can easily be determined in the case that the page has a **Group** dictionary that specifies a CIE-based color space.

- Otherwise, flatten the colors to some assumed device color space with predetermined calibration. In the generated PostScript output, paint the flattened colors in a CIE-based color space having that calibration.

During conversion to PostScript, a decision must also be made about the set of available spot colors to assume. This is because the choice of spot colorant versus alternate color space affects the flattened results of the process colors. (This is unlike the situation with strictly opaque painting, where the decision can be deferred until the generated PostScript is executed.)

12 Overprinting, Erasing, and Transparency

This section provides details of the overprinting and erasing rules in PDF. The information here duplicates material that can be found elsewhere. However, the rules are complex and interact with transparency in non-obvious ways, so it is useful to have the information collected in one place and the results spelled out for every case. This material should be read in conjunction with Section 7.2, “Spot Colors,” and Section 7.3, “Overprinting and Erasing.”

Table 14 shows the existing rules as defined in the opaque imaging model of PDF 1.3 (which is an extension of the PostScript imaging model). Table 15 shows the equivalent rules, respecified as a special **Compatible** blend mode in the transparency imaging model of PDF 1.4.

OP and **OPM** refer to the overprint flag and nonzero overprint mode control parameters in PDF. **OP** is equivalent to the **setoverprint** value in PostScript. There is no PostScript equivalent to **OPM**; it enables or disables the interpretation of

DeviceCMYK color values to specify componentwise overprinting, as described in Section 7.3, “Overprinting and Erasing.”

TABLE 14 Overprinting and erasing rules in PDF 1.3

Source color space	Affected color component	Effect on that color component		
		OP false	OP true, OPM 0	OP true, OPM 1
DeviceCMYK & specified directly & not image	C, M, Y, or K	Paint source	Paint source	Paint source if source ≠ 0 Don’t paint if source = 0
	Process colorant ≠ CMYK	Paint source	Paint source	Paint source
	Spot colorant	Paint 0	Don’t paint	Don’t paint
Any process color space (including other cases of DeviceCMYK)	Process colorant	Paint source	Paint source	Paint source
	Spot colorant	Paint 0	Don’t paint	Don’t paint
Separation or DeviceN	Process colorant	Paint 0	Don’t paint	Don’t paint
	Spot colorant named in source space	Paint source	Paint source	Paint source
	Spot colorant not named in source space	Paint 0	Don’t paint	Don’t paint

TABLE 15 Overprinting and erasing recast as blend modes in PDF 1.4

Source color space	Affected color component of group color space	Value of blend mode Compatible(*Cb*, *Cs*), expressed as tint		
		OP false	OP true, OPM 0	OP true, OPM 1
DeviceCMYK & specified directly & not image	C, M, Y, or K	*Cs*	*Cs*	*Cs* if *Cs* ≠ 0 *Cb* if *Cs* = 0
	Process color component ≠ CMYK	*Cs*	*Cs*	*Cs*
	Spot colorant	*Cs* (= 0)	*Cb*	*Cb*
Any process color space (including other cases of DeviceCMYK)	Process color component	*Cs*	*Cs*	*Cs*
	Spot colorant	*Cs* (= 0)	*Cb*	*Cb*
Separation or DeviceN	Process color component	*Cs* (= 0)	*Cb*	*Cb*
	Spot colorant named in source space	*Cs*	*Cs*	*Cs*
	Spot colorant not named in source space	*Cs* (= 0)	*Cb*	*Cb*
Source is a group (not an elementary object)	All color components	*Cs*	*Cs*	*Cs*

In these tables, color component values are represented as subtractive tint values, because one ordinarily thinks of overprinting inks, not additive light values. The **Compatible** blend mode is described as if it took subtractive arguments and returned subtractive results. However, in reality, the **Compatible** blend mode (like all blend modes) treats color components as additive values; subtractive components must be complemented before and after application of the blend mode.

Please note an important difference between the two tables. In Table 14, the process color components being discussed are the actual device colorants—the color components of the native color space of the output device (**DeviceRGB**, **DeviceCMYK**, or **DeviceGray**). In Table 15, the process color components are those of the group's color space, which is not necessarily the same as that of the output device (and can even be something like **CalRGB** or **ICCBased**). For this reason, the process color components of the group color space cannot be treated as if they were spot colors (in a **Separation** or **DeviceN** color space); see Section 7.2, "Spot Colors."

This difference between PDF 1.3 and PDF 1.4 overprinting and erasing rules arises only within a transparency group, not when painting directly on the page. Since PDF 1.3 doesn't have transparency groups to begin with, this difference is not considered to be an incompatibility. (There is no difference in the treatment of spot color components.)

Table 15 has one additional row at the bottom. It applies when painting an object that itself is a transparency group instead of an elementary object (fill, stroke, text, image). As explained in Section 7.2, "Spot Colors," a group is considered to paint every color component, both process and spot. Color components that were not explicitly painted by any object in the group have an additive color value of 1 (subtractive tint 0).

Since no information is retained about which components were actually painted within the group, compatible overprinting is not possible in this case. The **Compatible** blend mode reverts to **Normal**, with no consideration of the **OP** and **OPM** parameters. (Note that a transparency-aware application can choose a more suitable blend mode, such as **Darken**, if it desires to produce an effect similar to overprinting.)

13 Revision History

November 30, 2000

Changed the semantics of the page group. The compositing formula for the page group is now applied only if the page is imposed on output media, not if it is placed on other content by an aggregating application. The formula itself is changed to isolate the page group from the backdrop. If a page object has a group-attributes dictionary, it specifies attributes of the page group, rather than

being a shorthand for specifying a subsidiary transparency-group XObject. The interpretation of the transparency group attributes for a page group is slightly changed.

Clarified the kinds of **ICCBased** color spaces that are permitted as group color spaces.