Masked Images

Adobe® Developers Association

10 October 1997

Technical Note #5601

LanguageLevel 3

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Eastern Regional Office
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120

Adobe Systems Europe Limited
Adobe House, Mid New Cultins
Edinburgh EH11 4DU
Scotland, United Kingdom
+44-131-453-2211

Adobe Systems Japan
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150 Japan
+81-3-5423-8100

PN LPS5601

Adobe Systems Incorporated

Contents

Adobe Systems Incorporated

Figures

Adobe Systems Incorporated

Tables

Adobe Systems Incorporated

Examples

Adobe Systems Incorporated

Preface

This Document

This is the original release for *Masked Images*, a document that provides a detailed description of masked images, a LanguageLevel 3 feature of the PostScript® language that enables a developer to create and use pixel-based masks for various types of images.

Intended Audience

This document is written for software developers who are interested in learning about masked images or adding masked images capabilities to an application that supports PostScript display or printing devices.

It is assumed that the developer has a strong background in image editing and page layout applications. This knowledge will help in the understanding of the pixel-based masks and the methods for supplying mask and image data to the application.

Organization of This Document

Section 1, “Masked Images,” gives an overview of the LanguageLevel 3 feature and all of its parts. A comparison of current and previous methods of masking images is made as well as the uses for, and benefits of, this feature in a PostScript language environment.

Section 2, “Implementing Masked Images,” defines the PostScript extensions for masked images. Each image mask type or method is described in detail. Several examples defining dictionary parameters (keys) are included as well as several workable code samples for each of the supported image mask types. The examples shown include use of image dictionaries and the **image** operator.

Section 3, “Masked Images Tips and Techniques,” gives helpful information on using masked images and selecting the best mask type for specific application needs.

Related Publications

Supplement: PostScript Language Reference Manual (LanguageLevel 3 Specification and Adobe PostScript 3™ Version 3010 Product Supplement), available from the Adobe® Developers Association, describes the formal extensions to the PostScript language that have occurred since the publication of the PostScript Language Reference Manual, Second Edition. This supplement also includes all LanguageLevel 3 extensions available in version 3010.

PostScript Language Reference Manual, Second Edition (Reading, MA: Addison-Wesley, 1991) is the developer's reference manual for the PostScript language. It describes the syntax and semantics of the language, the imaging model, and the effects of the graphical operators.

Statement of Liability

THIS PUBLICATION AND THE INFORMATION HEREIN IS FURNISHED AS IS, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY ADOBE SYSTEMS INCORPORATED. ADOBE SYSTEMS INCORPORATED ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES, MAKES NO WARRANTIES OF ANY KIND (EXPRESS, IMPLIED, OR STATUTORY) WITH RESPECT TO THIS PUBLICATION, AND EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSES, AND NONINFRINGMENT OF THIRD-PARTY RIGHTS.

Masked Images

1 Masked Images

1.1 Overview of Masked Images

Within the graphic arts industry, it is common to take an image (such as a photograph) from an image editing application, mask out a portion of the background, and then place the cropped image on a different background. The new image is often transferred to and used in a page layout application.

In previous levels of the PostScript language, the process of *masking* is commonly done by drawing a clipping path between the pixels of the source image. An image or other graphic must be painted within this path. Such paths often need many small line segments to produce. If the clipping path is very complex, such as what is needed for a tree and its leaves, or if there is more than one clipping path, such as for a forest of these trees, a **limitcheck** error might result when an attempt is made to save or store the clipping path or print it. It is also possible to run out of memory if the graphics state containing the clipping path or paths is saved.

In LanguageLevel 3, it is now possible to place images on the page without this concern for the constraints of the clipping path. It is now possible to indicate which individual pixels in an image will be painted and which will be masked. A *masked image* contains both the source data used in an ordinary image plus mask data which determines which image pixels will be painted. This assumes a 1-to-1 pixel interleave; in general, the pixels of mask data define which areas of an image are to be painted, where area can be as small as less than a pixel in size.

The PostScript language definition of the **image** operator and image dictionary has been extended to allow for two new image types, **ImageType 3** and **ImageType 4**, which represent the two supported image mask types (methods).

ImageType 3 masked images combine a regular sampled image with an explicit mask. The mask is treated essentially the same as the source data of the **imagemask** operator. It indicates the places on the page that are painted

and those that are masked (left unchanged). The unmasked portions are painted with the corresponding portions of the sampled image; the masked portions are not painted.

The description of a regular image contains *N* color components per pixel (where *N* depends on the current color space); the description of a masked image contains *N* color components plus one mask component per pixel. The mask data may be included with the image data or provided separately. The image and the mask must be coincident on the page (overlay one another); that is, they must be in the same position and approximately the same size (plus or minus a pixel-width). They do not need to be the same resolution.

ImageType 4 masked images are defined by specifying a color or range of colors. Pixels in the image that match this color or are within the range of colors are not painted, allowing the background image to show through. This technique is often called Chroma-key masking.

1.2 Benefits of Using Masked Images

There are several benefits in using masked images over older methods of providing masks:

- Masks can be represented and processed more efficiently. Complex and/or potentially large clipping paths are no longer required for masking.
- Masked images benefits all applications that edit images or place images (page layout) by improving performance and printing reliability.
- Image data is now controllable on a pixel-by-pixel basis, a scan line basis, or on a selected color basis.

2 Implementing Masked Images

Section 2.1, “Image Operator” describes the **image** operator and how it can be used to paint masked images. Section 2.2, “Image Dictionaries” through Section 2.5, “ImageType 4 Image Dictionary” covers the definition and use of image dictionaries for the individual image mask types and interleave types. Also included in Section 2.3 is a discussion of samples in image data.

Complete descriptions of all of the PostScript language extensions for masked images can be found in the *Supplement: PostScript Language Reference Manual*.

2.1 Image Operator

The **image** operator has been extended to render standard and masked images. Standard images are still rendered using an **ImageType 1** image dictionary as input to the **image** operator. A masked image is rendered by using an **ImageType 3** or **ImageType 4** image dictionary as input to the **image** operator.

Note *Examples using the **image** operator can be found in Sections 2.4 and 2.5 of this document.*

For more information on the **image** operator for standard images, see Sections 4.10 and 8.2 of the *PostScript Language Reference Manual, Second Edition*.

2.2 Image Dictionaries

Two new image types can be used with the **image** operator to produce masked images:

- **ImageType 3**: This dictionary contains two subdictionaries that describe the image and mask data. These are the **DataDict** and **MaskDict** dictionaries, respectively. Each of these is defined using an **ImageType 1** image dictionary.

  How the mask data is provided is specified by the **InterleaveType** key in the **ImageType 3** dictionary. The behavior of the mask is determined by the entries in the **MaskDict** dictionary (see Table 2)

- **ImageType 4**: This dictionary has the same format as an **ImageType 1** image dictionary with the added key **MaskColor** which specifies the color or range of colors that will be used as a mask.

Note *There are two new instances of the implicit resource **ImageType**. These are 3 and 4, pertaining to **ImageType 3** and **ImageType 4** image dictionaries.*

2.3 Samples in Image Data

In the context of image data, a *sample* is a set of bits that together define a single pixel. The simplest sample value is a single bit that defines a pixel whose values are either 0 (black) or 1 (white). More complex samples have multiple bits per pixel and are used to define shades of gray or colors. For example, a sample with 24 bits of data could define a pixel in the RGB color space with 8 bits per color component (8 bits for red + 8 bits for green + 8 bits for blue). As another example, a sample with 8 bits of data could define a pixel in the Gray color space whose value is one of 256 shades of gray (black, white, and 254 intermediate shades of gray). A single pixel (or sample) of an image has a *depth* defined by the number of bits per pixel (or sample). Pixel or sample depth plays an important role in the context of how pixel samples are arranged to from an image composed of scan lines.

An image is composed of a number of scan lines or rows. The number of rows in an image defines the *height* of the image. Each scan line is composed of a number of samples. The number of samples in the scan line defines the *width* of the image. A complete image has a width consisting of *n* samples (columns) per scan line, a height consisting of *m* scan lines (rows), and a depth consisting of *d* bits per pixel. This can be represented by the formula

$n * m * d$

Multiple color component image data (for example, data for the RGB color space) for Type 1 images can be interleaved either at the sample level or the scan line level. For image data that is interleaved at the sample level, the **MultipleDataSources** key (see Section 4.10.5 of the *PostScript Language Reference Manual, Second Edition*) must be set to false. Figure 1 shows this data representation for image data in the RGB color space.

Figure 1 *Sample level interleaving for a Type 1 image in the RGB color space*

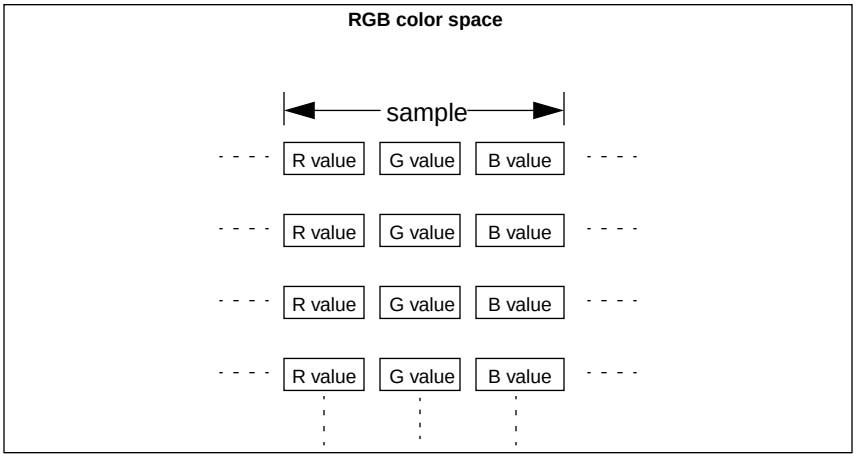
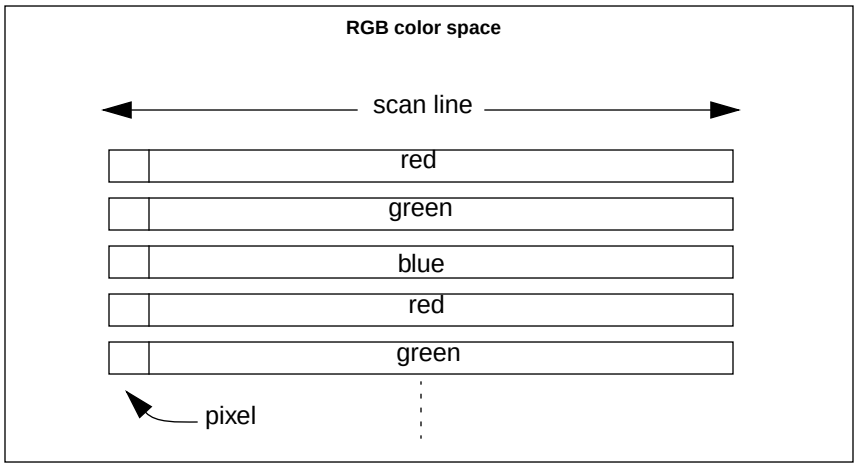


Image data that is interleaved at the scan line level is arranged so that one entire scan line of data is given for each color component. For example, in the RGB color space, there would be one scan line of red, followed by one scan line of green, followed by one scan line of blue, and so on. In this case, the **MultipleDataSources** key must be set to true and three separate procedures (one for each color component) are required to deliver the image data. shows this data representation for image data in RGB color space. Figure 2 shows this method.

Figure 2 *Scan line level interleaving for a Type 1 image in the RGB color space*



For image data that is interleaved at the scan line level, the color component of a given pixel represents a slice through the consecutive scan lines that contain the color components.

In LanguageLevel 3, the notion of the image sample has been maintained, and the two interleaving methods described above have been extended for use with the **ImageType 3** image dictionary for masked images. In addition, a new interleaving method has been added that assumes the image data and mask data come from separate sources and different data channels. For **ImageType 4** image dictionaries, the image data can be defined similarly to the way it is defined for Type 1 images.

2.4 ImageType 3 Image Dictionary

A Type 3 (masked) image requires three dictionaries:

- An **ImageType 3** image dictionary. This dictionary is the argument to the **image** operator.
- A subdictionary to define how the mask data is laid out. This dictionary is a modified image Type 1 dictionary and is referenced by the **MaskDict** key.
- A subdictionary to define how the image data is laid out. This dictionary is a modified image Type 1 dictionary and is referenced by the **DataDict** key.

Figure 3 shows the relationship and hierarchy of the dictionaries needed for **ImageType 3** images.

Figure 3 *Dictionaries used for ImageType 3 image masks*

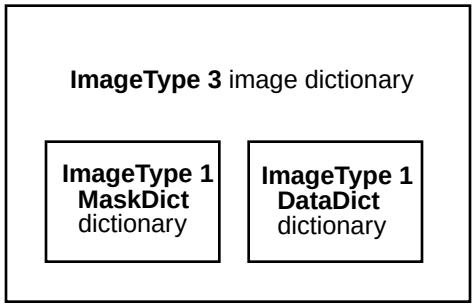


Table 1 lists the keys used to define an **ImageType 3** image dictionary. The keys are described in more detail below.

Table 1 *Keys for the ImageType 3 image dictionary*

Key	Type	
ImageType	integer	required
InterleaveType	integer	required
MaskDict	dictionary	required
DataDict	dictionary	required

ImageType must have a value of 3.

The entries in the **MaskDict** and **DataDict** dictionaries are described in Table 2 and Table 3, respectively. The **InterleaveType** key is discussed after the presentation of these two dictionaries and their associated keys (See “InterleaveType Key” on page 22).

Note *For more information on **ImageType 1** image dictionaries, see Section 4.10.5 of the PostScript Language Reference Manual, Second Edition.*

Note *Entries in the **MaskDict** dictionary are interpreted as if the dictionary were supplied to the **imagemask** operator, except as indicated in Table 2.*

Table 2 lists the keys used for defining the **MaskDict** dictionary. The keys are described in more detail below.

Table 2 *Keys for the MaskDict dictionary*

Key	Type	
ImageType	integer	required
Width	integer	required
Height	integer	required
ImageMatrix	array	required
MultipleDataSources	Boolean	optional
DataSource	various	required (see below)
BitsPerComponent	integer	required
Decode	array	required
Interpolate	Boolean	optional

ImageType must have a value of 1.

Width specifies the width of the mask in samples. If this is used with **InterleaveType 1**, then the value of **Width** must be equal to the value of **Width** defined in **DataDict** (see Table 3).

Height specifies the height of the mask. If this is used with **InterleaveType 1**, the value of **Height** must be equal to the value of **Height** defined in **DataDict** (see Table 3). If this is used with **InterleaveType 2,** the value of **Height** may differ from the value of **Height** in **DataDict** with the following restrictions: one must be an integral multiple of the other; that is, there may be multiple lines of mask data per line of image data or multiple lines of image data per line of mask data; the value of either mask height or image height must be unity.

ImageMatrix is an array of six numbers that defines a transformation from current user space to image source space. This matrix must be approximately equal to the value of **ImageMatrix** defined in **DataDict** (see Table 3) scaled by the difference in size of the mask and image data.

If the **MultipleDataSources** key is present in the dictionary, then its value must be set to false.

DataSource is required for **InterleaveType 3**. For the other two interleave types, it must not be present. **DataSource** specifies the data source for the mask data. It can be a string, a stream, a file, or a filter.

If **BitsPerComponent** is used with **InterleaveType 1**, its value must be one. Otherwise, the value of **BitsPerComponent** must be equal to the value of **BitsPerComponent** defined in **DataDict** (see Table 3).

The **Decode** array describes how to map mask sample values into the appropriate range of values. The length of this array must be two. If the array is [0 1], a mask sample value of 0 designates a painted sample, and a value of 1 designates an unpainted sample. If the array is [1 0], a mask sample value of 1 designates a place to be painted, and a value of 0 designates a place to be masked.

If the **Interpolate** key is present and true, then interpolation will be performed on the mask. The default value for **Interpolate** is false.

Note *Entries in the **DataDict** dictionary are interpreted as if they were entries for a **ImageType 1** image dictionary, except as indicated otherwise in Table 3*

Table 3 lists the keys used to define the **DataDict** dictionary. The keys are described in more detail below.

Table 3 *Keys for the DataDict dictionary*

Key	Type	
ImageType	integer	required
Width	integer	required
Height	integer	required
ImageMatrix	array	required
MultipleDataSources	Boolean	optional
DataSource	various	required
BitsPerComponent	integer	required
Decode	array	required
Interpolate	Boolean	optional

ImageType must have a value of 1.

Width specifies the width of the source image in samples.

Height specifies the height of the source image in samples.

ImageMatrix is an array of six numbers that defines a transformation from current user space to image source space.

MultipleDataSources specifies whether or not the image data is provided through multiple data sources. If the value is false, there is only one data source. If the value is true and **InterleaveType 3** is being used, multiple data sources are used, one per color component, and one for the mask data.

DataSource specifies the data source for the image data. If it is used with **InterleaveType 1** or **2**, the mask data is interleaved with the image data either at the sample level or the scan line level. If it is used with **InterleaveType 3**, the data source for the mask is separate from the data source for the image data (see the *PostScript Language Reference Manual, Second Edition*, Section 4.10.5).

BitsPerComponent specifies the number of bits used to represent each color component. The value of this key can be 1, 2, 4, 8, or 12. The number of bits is the same for each color component.

The **Decode** array specifies how to map image sample values into the range of values appropriate for the current color space. The length of this array must be exactly twice the number of color components in the current color space.

The default value for the **Interpolate** key is false. If the key is present and its value is true, then image interpolation will be performed.

InterleaveType Key

The **InterleaveType** key in the **ImageType 3** image dictionary specifies if and how the image data and mask data (as defined in the **DataDict** and **MaskDict** dictionaries above) are interleaved, or stored together. The **InterleaveType** key can have one of three values – 1, 2, or 3.

InterleaveType 1

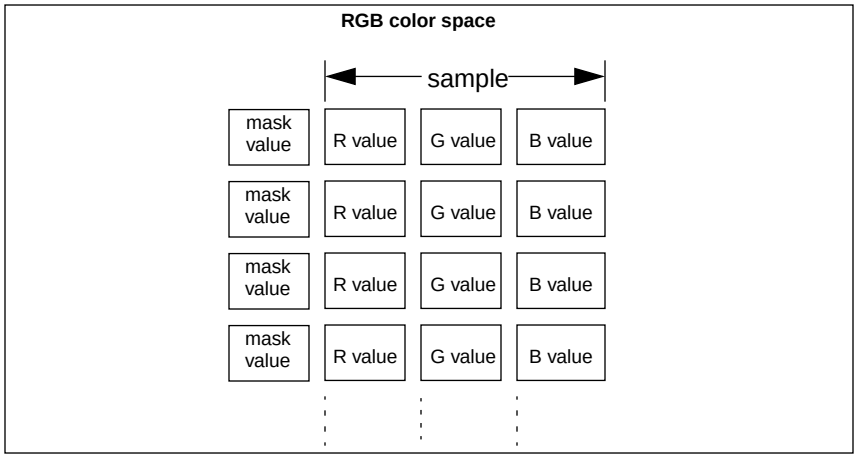
A value of 1 indicates that the image data and the mask data are *component (or pixel) interleaved*. This means that the mask data is interleaved with the image data on a per-sample basis, with the mask data presented first.

The number of bits per component in the mask (from the key **BitsPerComponent** contained in **MaskDict**) must be the same as the number of bits per component of the image source (from the key **BitsPerComponent** contained in **DataDict**).

The value of the mask component should be either all zeros or all ones. If any other value(s) is/are specified, it/they will be treated as a one.

Figure 4 shows how data is component interleaved in the RGB color space.

Figure 4 *Data representation for InterleaveType 1 in the RGB color space*



Example 1 shows an **ImageType 3** mask using **InterleaveType 1**.

Note *A complete PostScript language file containing this example accompanies this document.*

Example 1 *ImageType 3 image using InterleaveType 1*

```
%MASKIM31.PS
%This example illustrates Type 3 Masked Images with
%Type 1 interleave and using /ReusableStreamDecode
%Note that the /ReusableStreamDecode filter is optional
%for InterleaveType 1. The data could be put in-line
%right after the call to the image operator.
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
ff99ffff ff99ffff ff99ffff ff99ffff ff99ffff ff99ffff
...% image and mask data
>
/datastream exch def
/inch {72 mul} def
/DeviceRGB setcolorspace
%DataDict
/ImageDataDictionary 8 dict def
ImageDataDictionary begin
    /ImageType 1 def
    /Width 317 def
    /Height 299 def
    /BitsPerComponent 8 def
    /DataSource datastream def
    /MultipleDataSources false def
    /ImageMatrix [317 0 0 299 0 0] def
    /Decode [0 1 0 1 0 1] def
end
%MaskDict
/ImageMaskDictionary 8 dict def
ImageMaskDictionary begin
    /ImageType 1 def
    /Width 317 def
    /Height 299 def
    /BitsPerComponent 8 def
    /MultipleDataSources false def
    /ImageMatrix [317 0 0 299 0 0] def
    /Decode [0 1] def
end
/MaskedImageDictionary 7 dict def
MaskedImageDictionary begin
    /ImageType 3 def
    /InterleaveType 1 def
    /MaskDict ImageMaskDictionary def
    /DataDict ImageDataDictionary def
end
%Reset reusable stream and do initial image
...% add code here
%Reset reusable stream, invert Decode and do next image
...% add code here
showpage
```

InterleaveType 2

An **InterleaveType** value of two indicates that the image data and the mask data are *scan line interleaved*. This means that the mask data is interleaved with image data at the boundary between two scan lines, with the mask data presented first. The mask and the image sample scan lines must be padded to byte boundaries separately. The mask data will always be one bit-per-pixel regardless of the image sample depth.

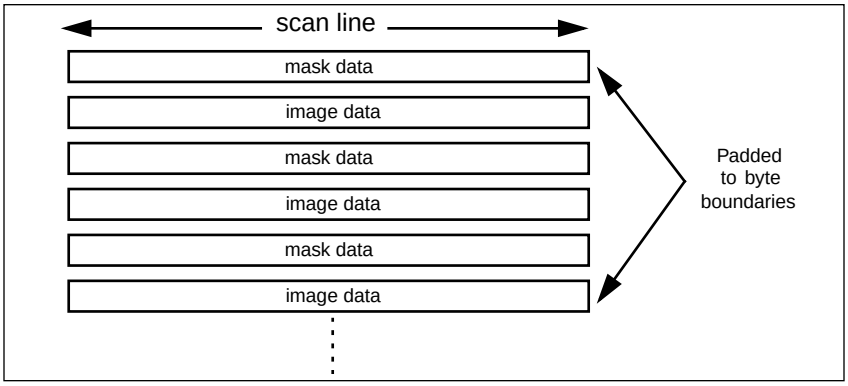
The height of the mask (specified by the **Height** key in **MaskDict**) may differ from the height of the image data (specified by the **Height** key in **DataDict**), with the following restrictions: one must be an integral multiple of the other; that is, there may be multiple lines of mask data per line of image data or multiple lines of image data per line of mask data. It is not possible to group multiple lines of both types of data. For example, three lines of mask data per one line of image data is supported, but three lines of mask data per two lines of image data is not. The value of either the mask height or the image height must be unity.

The smaller of either the mask **Height** or image **Height** values defines the number of *interleave blocks*. An interleave block consists of either one row of mask data followed by one or more rows of image data, or, one or more rows of mask data followed by one row of image source data. All interleave blocks must have the same number of scan lines of each data type. The relationship between the **Height** values in each dictionary will determine the format of the interleave block.

The width values of each dictionary are arbitrary.

Figure 5 shows how data is scan line interleaved.

Figure 5 *Data representation for InterleaveType 2*



Example 2shows an **ImageType 3** mask using **InterleaveType 2**.

Note *A complete PostScript language file containing this example accompanies this document.*

Example 2 *ImageType 3 image using InterleaveType 2*

```
% This example illustrates Type 3 Masked Images with
% Type 2 interleave and using /ReusableStreamDecode
%Note that the /ReusableStreamDecode filter is optional
%for InterleaveType 2. The data could be put in-line
%right after the call to the image operator.
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
FFFFFFFFFFFFFFFFFFFFFF87FFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFF8
...%image and mask data
ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff
99ffff99ffff
>
/datastream exch def
/inch {72 mul} def
/DeviceRGB setcolorspace
/ImageDataDictionary 8 dict def
ImageDataDictionary begin
...%Similar to Example 1
end
/ImageMaskDictionary 8 dict def
ImageMaskDictionary begin
...%Same as for Example 1
end
/MaskedImageDictionary 7 dict def
MaskedImageDictionary begin
    /ImageType 3 def
    /InterleaveType 2 def
    /MaskDict ImageMaskDictionary def
    /DataDict ImageDataDictionary def
end
%Reset reusable stream and do initial image
...% add code here
%Reset reusable stream, invert Decode and do next image
...% add code here
showpage
```

InterleaveType 3

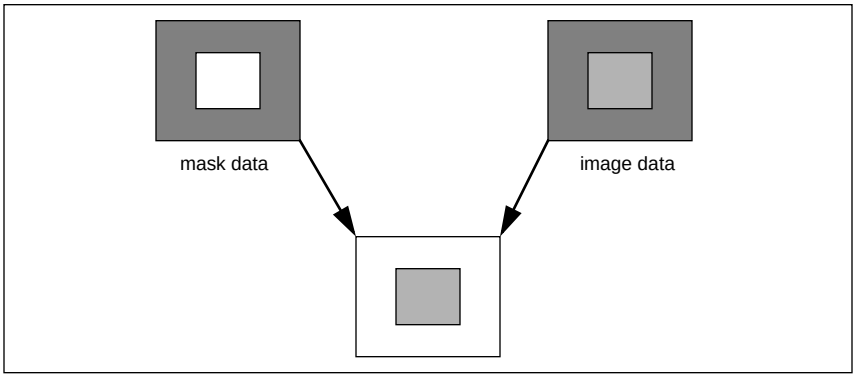
Note ***InterleaveType 3*** *is intended for high-end or disk-based printing systems because of the need for extra space or memory to hold the separate mask and image data sources.*

An **InterleaveType** value of type three indicates that the image data and the mask data are provided through separate sources and are sent on different data channels. The mask data source is defined in the **MaskDict** subdictionary of the **ImageType 3** image dictionary. The image data source is defined in the **DataDict** subdictionary of the **ImageType 3** image dictionary.

The height and width of the mask data are independent of the height and width of the image source data, but the mask must have the same orientation and placement as the image.

Figure 6 shows a data representation when the image data and mask data come from separate sources and data channels.

Figure 6 *Data representation for InterleaveType 3*



The keys of the **ImageType 3** image dictionary, **MaskDict** dictionary, and **DataDict** dictionary are also described in Section 4.3 of the *Supplement: PostScript Language Reference Manual*.

Example 3 shows an **ImageType 3** mask using **InterleaveType 3**.

Note *A complete PostScript language file containing this example accompanies this document.*

Note *Complete information on* ***ImageType 3*** *image dictionaries can be found in Section 4.3 of the Supplement: PostScript Language Reference Manual.*

Example 3 *ImageType 3 image using InterleaveType 3*

```
% This example illustrates Type 3 Masked Images with
% Type 3 interleave and using /ReusableStreamDecode.
% The common practice is to actually have only the mask
% data in a reusable stream.
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
FFFFFFFFFFFFFFFFFFFFFF87FFFFFFFFFFFFFFFFFFFFFFFF
...% mask data
FFFFFFFFFFFF8
>
/maskstream exch def
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff
...% image data
99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff
>
/datastream exch def
/inch {72 mul} def
/DeviceRGB setcolorspace
/ImageDataDictionary 8 dict def
ImageDataDictionary begin
...% same as for Example 1
end
/ImageMaskDictionary 8 dict def
ImageMaskDictionary begin
...% same as for Example 1
end
/MaskedImageDictionary 7 dict def
MaskedImageDictionary begin
    /ImageType 3 def
    /InterleaveType 3 def
    /MaskDict ImageMaskDictionary def
    /DataDict ImageDataDictionary def
end
%Reset mask and data streams and do initial image
...% add code here
%Reset mask and data streams, invert Decode,
%and do next image
...% add code here
showpage
```

2.5 ImageType 4 Image Dictionary

An **ImageType 4** image dictionary is used for defining *Chroma-key* images. This allows for a given color, or a range of colors, to be defined as the mask value. In other words, it defines the color or colors that are not printed, which allows the background image or color to be revealed.

Note *Entries for the **ImageType 4** image dictionary are interpreted as if they were entries for a **ImageType 1** image dictionary, except as indicted otherwise in the following table.*

Note *The **ImageType 4** image dictionary can not be used with the **imagemask** operator.*

The keys defining an **ImageType 4** image dictionary are listed in Table 4 and described in more detail, below. All of these keys are the same as those for the **ImageType 1** image dictionary, except for the new key **MaskColor**.

Table 4 *Keys for the ImageType 4 image dictionary*

Key	**Type**	
ImageType	integer	required
MaskColor	integer array	required
Width	integer	required
Height	integer	required
ImageMatrix	array	required
MultipleDataSources	Boolean	optional
DataSource	various	required
BitsPerComponent	integer	required
Decode	array	required
Interpolate	Boolean	optional

The value of the **ImageType** key must be 4.

The **Width** key specifies the width of the source image in samples.

The **Height** key specifies the height of the source image in samples.

ImageMatrix is an array of six numbers that specify a transformation from current user space to image source space.

The **MaskColor** key is an array of values corresponding to the source representation of the color value that will be masked. These values must be specified in one of two ways:

- One value is given for each source color component. The **Indexed**, **DeviceGray**, and **Separation** color spaces need only one value.

- A pair of values is given for each source color component. In this case, an image sample is considered to match the mask color if each source color component lies within each pair (range) of values. Because each source color component must be tested individually, this option is slower than the one value per source color component specification.

Values are checked against the incoming data samples as they are read. No conversion or decoding is done, except for the decompressing of any compressed data.

The **MultipleDataSources** key specifies whether or not the image data is provided through more than one source. If the key is present and true, the image data is provided through multiple data sources, one per color component. If the key is false, the image data for all color components is packed into one data stream, interleaved on a per-sample basis.

DataSource specifies the data source for the image data. Its definition is covered in the *PostScript Language Reference Manual, Second Edition*.

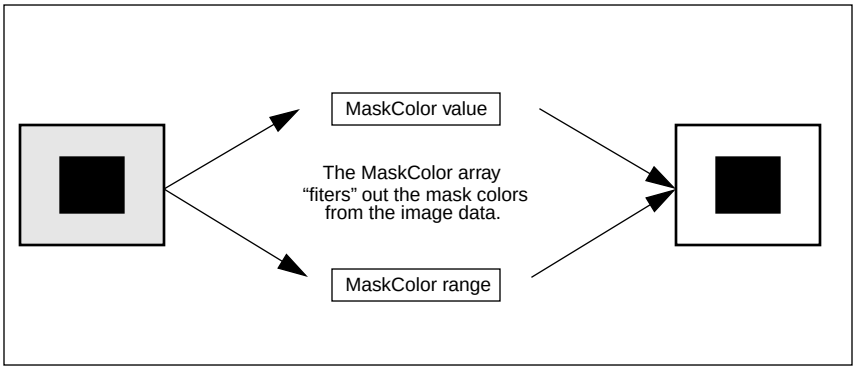
The **BitsPerComponent** key specifies the number of bits used to represent each color component. The value of this key can be 1, 2, 4, 8, or 12. The number of bits is the same for each color component.

The **Decode** array specifies how to map image sample values into the range of values appropriate for the current color space. The length of this array must be exactly twice the number of color components in the current color space.

The **Interpolate** key specifies whether or not interpolation is to be performed.If the value is set to true, then image interpolation will be performed. The value should be left as the default of false to prevent visual artifacts in the image.

Figure 7 shows how the image data for an **ImageType 4** image dictionary is filtered for a **MaskColor** range or value.

Figure 7 *Image data filtered by a MaskColor range or value*



Note *Complete information on **ImageType 4** image dictionaries can be found in Section 4.3 of the Supplement: PostScript Language Reference Manual.*

Example 4 shows an **ImageType 4** image mask.

Note *A complete PostScript language file containing this example accompanies this document.*

Example 4 *ImageType 4 image*

```
%MASKIM4.PS
%This example is an illustration of a masked image
%using a Type 4 image dictionary --- that is,
%a Chroma-key mask. Also uses /ReusableStreamDecode.
%Note that reusable streams are not needed for
%ImageType 4 image masks.
/inch {72 mul} def% define inch procedure
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff
...% image and mask data
99ffff99ffff99ffff99ffff99ffff99ffff99ffff99ffff
>
/datastream exch def
/ImageDictionary 7 dict def
ImageDictionary begin
    /ImageType 4 def
    /MaskColor [16#99 16#FF 16#FF] def
    /Width 317 def
    /Height 299 def
    /BitsPerComponent 8 def
    /DataSource datastream def
    /MultipleDataSources false def
    /ImageMatrix [317 0 0 299 0 0] def
    /Decode [0 1 0 1 0 1] def
end
%Reset the stream and place the first image...
...% add code here
%Reset the stream and place the second example...
...% add code here
%Reset the stream and place the third example...
...% add code here
showpage
```

3 Tips and Techniques

3.1 Using ImageType 3 image Dictionaries

ImageType 3 (mask component based images) is best used when detailed, pixel-by-pixel control is required over which pixels are printed. Use of this method, though, causes a substantial increase in the size of the data stream. For example, an RGB image of 8 bits per component (24 bits per pixel) using **InterleaveType 1** requires 32 bits per pixel. This is a 33% increase in the size of the data needed for the image.

3.2 Using the Various Interleave Types

With **InterleaveType 1**, the **BitsPerComponent** of the mask must be equal to the **BitsPerComponent** of the image data, leading to a potentially substantial increase in the overall data requirements for the image.

InterleaveType 2 and **3** are best used if the mask data can be generated either at the scan line level or as a separate data source. With these two types, each pixel of the mask is expressed as a single bit, thereby reducing the overall amount of data needed for the image.

3.3 Using ImageType 4 image Dictionaries

ImageType 4 (Chroma-key) is best used when there is a well-defined background color (or range of colors) in the image that are to be masked. The main advantage of using **ImageType 4** masks is that the mask data (a color or range of colors) is of minimal size compared to the image data and does not add a substantial amount to the size of the data needed for the image.

3.4 MaskColor Values or Ranges for ImageType 4

A single **MaskColor** value can be used when the background color of an image is uniform. In practice, this situation is unusual, though, and is normally only obtained when the background has been defined using a flat screen hue/color, such as is defined in an application. The more likely case it that the background is less flat and more complex (as in the background of a photograph).

A **MaskColor** range should be used when the background is more complex; that is, when it contains fluctuations, highlights, or shadows. The **MaskColor** range, then, defines all of the colors that comprise the background. Fluctuating backgrounds often result when images are scanned into an application.

3.5 Use of the MultipleDataSources Key

The **MultipleDataSources** key controls only the manner in which the image data is interleaved (see Sections 2.3 and 2.4 for more information). It does not control the way in which mask data interleaves with image data. For **ImageType 4** image dictionaries, there is no mask data to interleave with image data (although there is mask information in the form of a color value or range of color values). For **ImageType 3** image dictionaries, the **InterleaveType** key is used to control how the mask data and image data are interleaved (see the second half of Section 2.4 for more information).

Adobe Systems Incorporated

Index