Adobe® PostScript®

Open Prepress Interface (OPI) Specification – Version 2.0

Technical Note #5660

19 January 2000

Contents

Open Prepress Interface (OPI) Version 2.0

1 Introduction

The Open Prepress Interface (OPI) is a collection of PostScript® language conventions that allow low-resolution proxy images to be used for page layout. The high-resolution versions of the images are automatically substituted later by the image substitution server. Both desktop applications and electronic prepress systems can use OPI to minimize network traffic and image storage requirements.

PostScript language files containing OPI 2.0 comments must adhere to the Document Structuring Conventions, version 3.0 or later. See Appendix G, "Document Structuring Conventions—Version 3.0," in the *PostScript Language Reference Manual, Second Edition*, or Adobe Technical Note #5001, *Document Structuring Conventions, Version 3.0.* In this specification, the abbreviation "DSC" is used to refer to the Document Structuring Conventions.

The Backus-Naur form (BNF) of notation, used in the DSC 3.0 document, is also used in this specification.

1.1 Differences from OPI 1.3

OPI 2.0 introduces the need for the OPI producer to establish the graphics state. For information on setting the graphics state, see "<set the graphics state>" on page 14.

The other major difference from OPI 1.3 is the requirement for the comments to appear in a specific order. Section 2, "OPI 2.0 Comments," lists all comments in the order in which they must appear in a document. Additional differences are list in section 8, "Change History."

1.2 Terminology

Bitmap: A 1-bit deep image, typically specified using the PostScript operator imagemask.

Elementary DSC data type: An elementary or base type is a terminating expression that does not reference any other tokens and is considered to be a base on which other expressions are built. See Technical Note #5001, "PostScript Language Document Structuring Conventions Specification, Version 3.0."

Grayscale image: A monochrome image that is deeper than 1 bit per pixel (typically 8 bits).

Main image: The high-resolution version of the image, as it was scanned or otherwise acquired, from which the proxy image is created.

OPI: for the purpose of this document, "OPI" will only refer to the PostScript language convention associated with Adobe OPI version 1.3 and Adobe OPI version 2.0.

OPI Producer: An application that writes OPI comments—typically, a page layout program such as Adobe® InDesign™ or Adobe PageMaker® Plus.

OPI Consumer: An application that reads OPI comments and (usually) inserts high-resolution image data into the PostScript language stream. This is typically an image substitution server, but other prepress applications can also be OPI Consumers.

An application can be both an OPI Producer and an OPI Consumer. OPI Consumers should generally also be valid OPI Producers if they wish to write out an altered PostScript language stream, since another OPI Consumer may need to do further processing of the stream. An application should not assume that there is no other application between it and the imagesetter or platesetter final output, unless it is absolutely certain that no other application or RIP will post-process the output.

OPI Server: this term is often used to mean a server that resolves an image substitution method. For the purposes of this document, "image substitution server" will be used instead.

OPI context: Everything between "%%BeginOPI: 2.0" and the matching %%EndOPI comment, inclusive. OPI contexts may be included in EPS files that may be incorporated into other PostScript language streams.

PDF: Portable Document Format; a format that allows compact representation of graphics and text, and which is portable across multiple platforms.

Proxy image: The lower-resolution version of the image that is imported into a page layout program or other OPI Producer.

1.3   References

For the following references, all Adobe Technical Note documents are available at:

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

1. Portable Document Format (PDF) Reference Manual, Version 1.3.

2. *PostScript Language Reference, Third Edition*, Adobe Systems Incorporated.

3. Adobe Technical Note #5001, "PostScript Language Document Structuring Conventions Specification, Version 3.0." The name "Document Structuring Conventions" is sometimes abbreviated to "DSC."

4. Adobe Technical Note #5620, "Portable Job Ticket Format – Version 1.1."

5. Adobe Technical Note #5044, "Color Separation Conventions for PostScript Language Programs."

6. TIFF Revision 6.0, Aldus Corporation, June 3, 1992.

2 OPI 2.0 Comments

The following OPI comments are listed in the order they must appear in the file.

%%BeginOPI: 2.0

Required. Marks the beginning of an OPI 2.0 context.

%%Distilled   (no keywords)

%%Distilled is included in the stream at this point if and only if the original PostScript language stream was converted into a Portable Document Format (PDF) file, and then back to a PostScript language stream.

An OPI Consumer should use this information to reapply color to a colorized grayscale image, based on the OPI color comments, since the normal automatic color application environment created by the original OPI Producer is disabled when a PostScript code is converted into PDF, and then back again into PostScript code.

%%ImageFileName: **<filename>**

Required. %%ImageFileName: must be written by an OPI 2 Producer for every image in the stream that may be the target of OPI image substitution.

This comment records the full pathname of the low-resolution proxy image. *filename* is an elementary DSC type.

The low-resolution proxy image is typically a TIFF or EPS file. (Some applications require low-resolution proxy images to be TIFF files.)

%%MainImage: **<MainImageID>**

<MainImageID> ::= <textline>

Required if the low-resolution if the low-resolution proxy image is a TIFF file and the ImageID TIFF tag (#32781) exists.

MainImageID is the full pathname of the original, high-resolution file, or any other identifying string that uniquely identifies the main image. The MainImageID string is a textline, an elementary DSC type.

Note   *The high-resolution image is NOT required to be in TIFF format. It can be in any format that the OPI Consumer wishes to support, including Scitex CT, PDF, EPS, and Quark Desktop Color Separation (DCS) files.*

If the low-resolution proxy image is an EPS file instead of a TIFF file, the ID of the main image should be stored in a %%MainImage: comment in the EPS file, typically as part of an OPI context in the EPS file.

If the %%MainImage: comment does not exist, an OPI Consumer must do the best it can to find and link to the correct high-resolution image, using the name given in the %%ImageFileName: comment. If the image does not exist at the indicated location, the OPI Consumer can ask the operator to help re-link the image or use search path rules that have been set up ahead of time.

%%TIFFASCIITag: **<tagnumber> <tagtext>**

<tagnumber> ::= <uint> (TIFF tag number, in decimal)
<tagtext> ::= <textline> (contents of TIFF field)

An OPI 2.0 Producer must write this comment if the low-resolution proxy image is a TIFF file and if there are ASCII fields in the file. The Producer writes a separate comment for each TIFF ASCII field.

An OPI 2.0 Producer uses this comment convention to pass TIFF ASCII field values from the proxy image to an OPI Consumer.

tagnumber is the TIFF tag number. It is a 5-digit or less decimal number, and is not padded with zeros or spaces.

Examples:

%%TIFFASCIITag: 270 (Sunrise from Waikiki)

%%TIFFASCIITag: 306 (1995:02:14 13:55:59)

%%TIFFASCIITag: 316 (Apple PowerMac\250 8100)

%%TIFFASCIITag: 33432 (Copyright\251, Carl Stevensen, 1995. All rights reserved.)

%%TIFFASCIITag: 65535 (First substring.)

%%+ (Second substring.)

The value of %%TIFFASCIITag: is the value of the corresponding TIFF ASCII field. The OPI Producer must translate tagtext into textlines, an elementary DSC data type. So, for example, the Producer must check for special characters such as carriage return and line feed and translate them into the correct PostScript language escape sequences (see sections G.4.6 and 3.2.2 in the *PostScript Language Reference Manual, Second Edition*). The Producer must break comment lines longer than 255 characters into multiple lines using the '%%+' continuation convention.

Note that there may be more than one substring in a TIFF ASCII field. In the TIFF file, each substring is terminated with a null (zero) byte. When written out by the Producer, each substring after the first must start with a separate '%%+' comment line.

%%ImageDimensions: **<width> <height>**

<width> ::= <real>
<height> ::= <real>

%%ImageCropRect: **<left> <top> <right> <bottom>**

<left>::=<real>
<top>::=<real>
<right>::=<real>
<bottom>::=<real>

Optional, but if used, must be used together. %%ImageDimensions: and %%ImageCropRect: together specify the rectangular subset of the source image that is to be mapped onto the unit square (and thence onto the page), where (0,0) represents the upper left corner of the source image. See Figure 1.
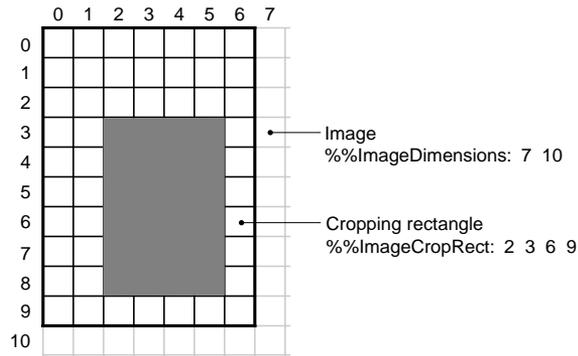
FIGURE 1 *Image Cropping*

If the image has not been cropped or placed inside of a frame by the OPI Producer, then these comments are optional, and the entire source image will be mapped onto the page.

width and *height* give the dimensions of the proxy image, in pixels, for image data, or any other convenient units such as points, for other types of graphics. The values of *width* and *height* do not matter; the only requirement is that

0 <= left < right <= width, and
0 <= top < bottom <= height

This information allows the OPI Consumer to improve printing performance by including only the part of the image data that is actually visible after the user of a page layout application has cropped it or placed it inside a frame, thereby reducing print time.

If sending less data is not possible, then the OPI Consumer must send the whole image, scale and translate the image such that only the cropped part of the image is mapped to the unit square.

In OPI 2.0 , the Producer is responsible for setting up a clipping path.

Note   *Any clipping from a TIFF clipping path must come after the rectangular clip path setting. Reversing the order greatly increases probability of limitcheck errors due to path complexity.*

Note   *An OPI 2.0 Consumer must not do its own clipping, or it will run a risk of causing limitcheck errors due to a complex TIFF clipping path that may be in effect at the time.*

See Figure 1.

%%ImageOverprint: **false | true**

Optional. The default is false.

Set to true if the image is to overprint underlying objects, and false if the image is to knock out underlying objects.

"Overprinting" means that any separations not included in the %%ImageInks: list (see below) are not to be erased in the area of the image. This could allow, for example, a black image to be printed on top of a light blue background, without erasing the inks that make up the light blue color. In the area of the image there may be cyan and magenta, as well as black ink. Overprinting can be useful for avoiding trapping problems, but can introduce unwanted colors if used indiscriminately.

"Knock out" means that any separations not included in the %%ImageInks: list are to be erased in the area of the image. In the example in the previous paragraph, imaging the black image would erase any inks that make up the light blue background, so that only black ink remains in the area of the image.

Note   *%%ImageOverprint: is applicable to bitmap and grayscale images only.*

%%ImageInks: **\<type> \<number_of_inks> \<name_of_ink_1> \<ink_1_level> ...\<name_of_ink_n> \<ink_n_level>**

\<type>::=\<monochrome | registration | full_color>
\<number of inks>::=\<uint>
\<name of ink n>::=\<text>
\<ink n level>::=\<real>

Optional. The default is:

   %%ImageInks: monochrome 1 (Black) 1.0

for black and white and grayscale images, and

   %%ImageInks: full_color

for RGB, CMYK, and CIE LAB images.

%%ImageInks: lists the inks that have been applied to the image, so that an OPI Consumer that can perform color separations knows whether or not data for this image should be included on a particular separation.

type can be monochrome, registration, or full_color (explained in detail below).

number_of_inks is the number of inks that make up the color that has been applied to the image.

name_of_ink_n is the name of the ink. This name must be a PostScript language string, as specified in the DSC specification.

ink_n_level is the amount of that ink to be applied to black (e.g., deep shadow) areas of the image. 1.0 represents full (100% dot) ink coverage, 0.0 represents none of that ink.

monochrome   For bitmap and grayscale images, use the type monochrome or registration.

If a "spot" (i.e. "custom") ink is applied to a monochrome image, just that ink will be in the ink list, so that the comment would be written as:

   %%ImageInks: monochrome 1 (Rose Red) 1.0

If a process color is applied to a monochrome image, then four inks make up the color, so the comment would be written as:

   %%ImageInks: monochrome 4 (Cyan) .23 (Magenta) .45 (Yellow) .10

    (Black) .02

If the image uses a single process ink, such as Black, the %%ImageInks: comment would list just Black:

    %%ImageInks: monochrome 1 (Black) 1.0

If this Black image is to be overprinted, so that ink layers other than Black are unaffected, be sure to write out %%ImageOverprint: true before the %%ImageInks: comment.

If a "hifi" (multi-ink) color is applied to a monochrome image, then there can be any number of inks. For example, we might have:

    %%ImageInks: monochrome 5 (Cyan) .10 (Magenta) .35 (Yellow) .50 (Black) .05 (Red) .60

A duotone can be created by applying two inks to a monochrome image. For example, we might have:

    %%ImageInks: monochrome 2 (Black) .50 (Sepia) .80

However, it is also generally desirable to adjust the transfer curves differently for each ink; such manipulations are not prohibited, but are beyond the scope of this specification.

Alternatively, a duotone, tritone, or quadtone can be specified as a multi-channel image, in which case the appropriate %%ImageInks: keyword would be full_color.

Generally, an OPI 2.0 Consumer does not need to worry about applying colors to monochrome images, since OPI 2.0 Producers must create the appropriate color environment by defining appropriate procedures and operators. All the OPI 2.0 Consumer needs to do is decide whether to insert image data or not, based on whether it is doing separations or composite printing, which separation is currently being printed, and which separations are affected by this image.

In particular, the OPI 2.0 Consumer should use the OPIimage operator for grayscale images. The OPI 2.0 Producer must define the OPIimage operator so that the correct color is automatically applied to the image. For more information on OPIimage, see section 3, "Requirements for OPI Producers."

The OPIimage operator uses the same arguments as the standard PostScript language image operator, either the 5-argument or dictionary form.

The inks listed in this comment must also be listed in the DSC comments %%DocumentCustomColors: or %%DocumentProcessColors:.

registration   If a monochrome image is to be printed on every separation, the comment would be written as:

%%ImageInks: registration

full_color   For RGB, CMYK, and CIE LAB 3- or 4-component color images, or any image containing more than 1 component, the %%ImageInks: comment would be:

%%ImageInks: full_color

Note   *A list of inks for multicomponent, "full_color" images are not specified, since the OPI Consumer typically wants to control the process of converting deep RGB and Lab images into the appropriate CMYK or DeviceN Color output color space for the current job.*

<set the graphics state>

Required. Before %%BeginIncludedImage, an OPI 2.0 Producer must set up the graphics state in such a way as to allow an OPI 2.0 Consumer to not have to deal with color, position, or clipping paths.

An OPI 2.0 Producer must first invoke the PostScript operator **save** or **gsave**, to allow the graphics state to be restored at the end of the OPI context.

Note   *Using **save/restore** has the advantage of allowing all VM used by the imaging process to be recovered. In particular, the string used for buffering image data may be quite large for wide images, and if there are multiple images on a page, this can start consuming a significant amount of VM. On the other hand, using **save/restore** increases the likelihood of hitting the PostScript interpreter limit on the number of save/restore contexts that can be nested. **gsave/grestore** has a higher limit than **save/restore** in most implementations, and is faster, but VM consumption could be a significant issue if there are many large images on a single page.*

In OPI 2.0 , the responsibility for image placement shifts from the Consumer to the Producer. After the save or gsave, an OPI 2.0 Producer must include PostScript language code that maps a unit square to the desired position, size, and rotation on the page. The Consumer is no longer

responsible for positioning the image. This coordinate mapping technique is a major difference between OPI 2.0 and previous versions of the OPI specification.

PostScript language code to set the color of bitmaps and grayscale images also goes here, as well as code to set up any clipping path that may be associated with the image. (See the Note on page 11 about clipping paths.)

%%BeginIncludedImage(no keywords)

Required. Must be written by OPI 2.0 Producers, whether proxy data is present or not.

%%BeginIncludedImage and %%EndIncludedImage bracket the actual image data and the call to the appropriate image procedure.

There must be no executable code between %%BeginIncludedImage and the last %%IncludedImageXXXX comment, %%IncludedImageQuality:.

%%IncludedImageDimensions:<pixelswide> <pixelshigh>

<pixelswide> ::= <real>
<pixelshigh> ::= <real>

Must be written by OPI 2.0 compliant Producers if image data is included between %%BeginIncludedImage and %%EndIncludedImage.

Do not write the %%IncludedImageDimensions: comment if no image data is included. Do not write the %%IncludedImageDimensions: comment if the Producer does not know what the pixel dimensions are, which will generally be the case when the proxy image is an EPS file.

pixelswide and pixelshigh give the width and height of the included image, in pixels, before cropping.

These values are used by an OPI Consumer to determine if the image data that is present in the PostScript language file is low-resolution—and hence needs to be replaced with a high-resolution version—or is already high-resolution and does not need to be replaced. This can easily be determined by comparing the %%IncludedImageDimensions: values with the dimensions of the high-resolution image that is about to be placed into the PostScript language stream.

The %%IncludedImageDimensions: values may not be the same as the %%ImageDimensions: values. For example, if a user chooses a "for position

only" print setting, the Producer may send only a subset of the placed image data to the printer.

%%IncludedImageQuality: <quality>

<quality> ::= <real>

Must be written by OPI 2.0 compliant Producers if image data is included between %%BeginIncludedImage and %%EndIncludedImage.

This is the last of the %%IncludedImageXXX comments. OPI 2.0 Consumers can expect to find image operands and image data after this comment.

Do not write the %%IncludedImageQuality: comment if no image data is included.

quality can have a value of 1.0, 2.0, or 3.0.

A *quality* value of 1.0 means that the included image data has definitely been subsampled, so that an OPI Consumer should make every attempt to substitute higher-resolution image data.

There are at least two conditions that should cause an OPI Producer to set a *quality* value of 1.0:

(1) The OPI Producer has subsampled the version of the image that was imported into it.

(2) The image imported by the OPI Producer is a TIFF file, the OPIProxy tag (#351) was present in the TIFF file, and the value of the OPIProxy tag is 1. This marks the image as a low-resolution proxy image.

A quality of 2.0 means that the included image data was the highest resolution that could be found by the OPI Producer, but a higher resolution version may exist elsewhere. An OPI Consumer should attempt to replace it with a higher-resolution version if possible, but not finding a higher-resolution version may not be a fatal error.

A quality of 3.0 means that the included image data is certain to be sufficient for final printing. An OPI Consumer should not replace the included image data with new image data.

An OPI Consumer should be prepared to do something reasonable even if the value for quality is something other than 1.0, 2.0, or 3.0.

<Image Operands & Data>

If image data is included, it goes here, along with a call to the desired form of the **image** operator and its operands.

The image matrix must be set to [w 0 0 −h 0 h], where w and h are the width and height of the image, in pixels. (This assumes that the visual top of the image is the first scan line. If the visual top of the image is the last scan line, use [w 0 0 h 0 h].) This causes the image to map itself onto the unit square. As mentioned above, the OPI Producer must have previously set up the appropriate transformation to map the unit square onto the correct place on the page.

Note   *In contrast to the OPI 1.3 conventions, this OPI 2.0 convention makes it unecessary for an OPI Consumer to know or care where the image goes on the page.*

If the low-resolution proxy image is an EPS file, PostScript language code to map the EPS file to the unit square goes here, before the EPS data. Code to map an EPS file onto the unit square might look something like:

[A 0 0 D E F] concat

where A = 1/(urx−llx), D = 1/(ury−lly), E = −llx/(urx−llx), F = −lly/(ury−lly),

and (llx,lly),(urx,ury) are the %%BoundingBox: coordinates of the EPS file.

Or, if it is preferred to think about it in two separate operations, and remembering that the concat operator pre-multiplies the new matrix with the CTM, the operations must be done in right-to-left order:

1/(urx−llx) 1/(ury−lly) scale

−llx −lly translate

Producers must not put anything between %%IncludedImageQuality: and %%EndIncludedImage that is intended to apply to a replaced high-resolution image, since everything between these two comments is removed by an OPI Consumer.

If image data is included in the file, the data and the call to the image operator must be surrounded by the %%Begin(End)Data comments; see Adobe Technical Note #5001, PostScript Language Document Structuring Conventions, Version 3.0. These comments allow the OPI Consumer to scan quickly to the end of the image data, and not get confused by trying to

scan binary data. These comments must not be written if no data is included. See section 5.1, "Example 1."

%%EndIncludedImage (no keywords)

Required, whether proxy data is present or not.

This marks the end of the OPI 2.0 included image data.

See %%BeginIncludedImage.

%%EndOPI (no keywords)

%%EndOPI must be written by OPI 2.0 Producers.

This marks the end of the OPI 2.0 context.

(See %%BeginOPI:)

Following %%EndOPI, the OPI Producer must call the PostScript operator restore, or grestore, to restore the interpreter state to its condition prior to the %%BeginOPI: comment.

3 Requirements for OPI Producers

An OPI Producer can be any program that either creates a PostScript language stream containing OPI comments, or adds or changes OPI comments in an existing stream. The initial OPI Producer is typically a page layout or illustration application, but it could also be an image substitution server that wishes to create EPS proxy images using OPI 2.0 comments.

The requirements for OPI 2.0 Producers have changed significantly from OPI 1.X requirements, to reduce the effort required to design and implement an OPI Consumer.

The PostScript language file created by an OPI 2.0 Producer must conform to the following requirements:

Color Separation Conventions

The color separation conventions provide guidelines for structure and content of PostScript language files for use in color separation environments. The current version does not require the use of PostScript LanguageLevel 2 separation facilities.

Prepare the graphics state

An OPI 2.0 Producer must set up a number of aspects of the graphics state before the %%BeginIncludedImage comment, so that the OPI Consumer does not have to deal with them. In particular, the OPI 2.0 Producer must:

- Set up a clipping path around the image.

- Set up the Current Transformation Matrix (CTM) so that a unit square gets mapped to the correct position, scale factor, skew, and rotation of the image on the page.

- Create PostScript language instructions that will apply the correct color to a monochrome image.

  In particular, a procedure named OPIimage must be defined by an OPI 2.0 Producer. The OPIimage procedure is used by both Producer and Consumer for all grayscale images. The parameters for OPIimage are the same as for the standard image operator, either the 5-argument or dictionary form..

- Create PostScript language instructions to set up a clipping path for a TIFF file that contains a clipping path. Use %ADBBeginImageClipPath and %ADBEndImageClipPath (no arguments) to encapsulate the clipping path, since an OPI Consumer may want to replace it with a higher-resolution version.

- Create PostScript language instructions to effect any changes made to the screen frequency or spot function of an image.

The OPI 2.0 Consumer does not have to worry about these aspects of the graphics state. The only obligation of the OPI Consumer is to find the high-resolution image, decide whether the image is of a higher resolution than any image data already in the PostScript language stream, and, if so, insert the high-resolution image data into the PostScript language stream, replacing any lower-resolution data, calling the OPIimage operator if the image data is grayscale.

OPI 2.0 versus OPI 1.X

It is up to the OPI Producer whether or not to write out both OPI 1.X and OPI 2.0 comments. It is possible to create a PostScript language stream that is readable by

both OPI 1.X and OPI 2.0 Consumers. See Example 2 below in the Examples section.

Composite versus separated PostScript language streams

OPI was originally designed for use with composite color PostScript files. However some prepress applications now use OPI with separated PostScript language streams, so that they can avoid dealing with the intricacies of producing LanguageLevel 1 separations from composite color PostScript language streams.

This can work, but there are some disadvantages to using separated streams with OPI:

• Separated streams are device dependent.

• Separated streams are larger than composite streams, since much of the stream is duplicated four or more times.

• A number of prepress technologies including Adobe PostScript 3™ and Adobe Extreme™ are optimized for composite PostScript or PDF files instead of separated streams, since a composite stream is smaller, the resulting display list is smaller, and the output frame buffer can be created in one pass. With the acceptance of composite files, additional functionality can be added to the RIP stage including in-RIP Trapping, color separations, CIP3, imposition, and late-stage editing.

It is beyond the scope of this document to try to codify and document existing practice for producing and manipulating separated streams, but if you do attempt to use separated files with OPI, here are a few things to watch out for:

Be sure to follow the existing de facto DSC comment convention for "%%PlateColor: colorname," so that the OPI consumer can know what the current separation is. The %%PlateColor: comment generally comes right after a %%Page: comment. For example:

```
 ...
 %%Page: 1 2
 %%PlateColor: Yellow
 %%BeginPageSetup
```

. . .

- Use the systemdict version of the image operator, or the separationimage color convention procedure, to image each separation of a CMYK image, since the color conventions allow a conforming color Producer to redefine the image operator to paint only on the Black plate.

See Section 2, "OPI 2.0 Comments" for further details on requirements for OPI Producers.

4 Requirements for OPI Consumers

An OPI Consumer can be any of the following: an OPI-capable print server such as the Extreme technology from Adobe, a high-end prepress workstation, or any software package that can read OPI comments and substitute high-resolution image data, such as Adobe InDesign or Adobe PageMaker Plus software.

OPI 2.0 Consumers have a much simpler task than in earlier versions of OPI. With OPI 2.0 , all of the work to prepare the graphic state environment for the image is provided by the OPI 2.0 Producer. The only remaining tasks for the Consumer are to locate the correct high-resolution image, separate it if necessary, and insert the image data into the PostScript language stream.

> *Note   In some PostScript language interpreter environments, it is possible and desirable to set up the image operator to read the image from a disk file, without explicitly inserting the image data into the stream. See Appendix B in this specification for more information.*

OPI 2.0 versus OPI 1.X

It is up to the OPI Consumer whether or not it wants to support both OPI 1.X and OPI 2.0 comments. Supporting only OPI 2.0 is much easier, since the OPI 2.0 Consumer can ignore all of the OPI 1.X comments.

Workflow position

It is never safe for an application to assume that it is the last one in a PostScript workflow. If your application replaces low-resolution image data with high-resolution image data, it is possible that another application farther downstream will replace your high-resolution data with even higher-resolution data. So be sure

to follow all the OPI rules and comment conventions when writing out a new PostScript language file—there may be another Consumer downstream.

Ignore unknown comments

An OPI Consumer should ignore comments that it does not recognize, so that it can continue to process documents that contain newer OPI comments. In this way, OPI can be expanded to meet industry demands without requiring that all OPI Consumers be immediately updated when a new revision of the specification is released.

It is possible that enumerated values may be added to existing OPI comments at some point in the future, so parsing software should be prepared to encounter unexpected values and respond appropriately, usually by using the default values for the comment. An example might be a new image data type for %%ImageInks:.

It is also possible that additional parameters may be added to the end of the argument list of an existing comment at some point. Parsing software should be prepared to encounter such additional parameters, and respond appropriately, usually by ignoring the additional parameters.

Special operators

If the %%Distilled comment is not present, the OPI Consumer should call the Producer-supplied OPIimage operator for all grayscale images. OPIimage automatically applies the correct color to the grayscale image.

Note   *OPI comments can be passed through Acrobat Distiller into the PDF file by selecting the* Preserve OPI *option in Distiller.*

If the %%Distilled comment is present, the OPI Consumer must construct its own procedures for applying the indicated process, spot, or multi-ink color to the grayscale image.

Consumers of separated PostScript language streams should use the *systemdict* **image** operator for separations of full color images. Just calling 'image,' without getting it from *systemdict*, may not do what you expect, since it is commonly redefined by color separation procedures.

5 Examples

5.1 Example 1

In this example a 5760x7200-pixel image has been cropped down to an 80x60-pixel subset of the image. A reduced-resolution version, sub-sampled down by a factor of 10, has actually been used for printing here, resulting 8x6 pixels of image data. (This is, of course, not representative of the amount of data in most real-world jobs.)

```
...application-specific PostScript language code begin a save context...
...application-specific code to set a clipping path around the image...
...application-specific code to set the color of monochrome images...
...optional application-specific code to apply changes to the lightness
   or contrast of the image...
...application-specific code to map the unit square onto the right place
   on the page, including any rotation, scale, skew, and mirror...
...optional application-specific code to set up a TIFF clipping path...
%%BeginOPI: 2.0
%%ImageFileName:   (Server:Disk1:941106:Carlsen:hi-res:transistor.tif)
%%ImageDimensions: 5760.0 7200.0
%%ImageCropRect: 30.0 20.0 110.0 80.0
%%ImageInks:  monochrome 1  (PANTONE 485 CV) 1.0
%%BeginIncludedImage
%   This is the section, between %%BeginIncludedImage and
%   %%EndIncludedImage, that an OPI 2.0 - compliant Consumer can
%   replace with higher-resolution image data.
%   If you replace it, don't forget to write out new
%   %%IncludedImageDimensions: and %%IncludedImageQuality: comments,
%   since you might not be the last OPI Consumer in the pipeline.
%%IncludedImageDimensions:  576   720
%%IncludedImageQuality:  1.0
8 6       % width and height of actual image data, in pixels (this image
is cropped)
/_h exch def
/_w exch def
/imbuf _w 7 add 8 idiv string def
_w _h true [_w 0 0 _h neg 0 _h] {currentfile imbuf readhexstring pop}
%%BeginData: 28 Hex Bytes
imagemask
FF
81
99
99
81
```

```
FF
%%EndData
%%EndIncludedImage
%%EndOPI
...application-specific code to restore the save context...
```

5.2 Example 2

Here is a small (16x16) grayscale image.

In this example, the code is backward-compatible with OPI 1.3 comments. Please
see the OPI 1.3 specification for details on the %ALD comments.

Note   *This example uses PostScript language comments beginning with "% " (percent/space) for*
the purposes of commenting on the OPI comments. Such comments would not appear in
the actual PostScript file.

```
...start save context...
...set up default user space...
%ALDImageFileName: MyDisk:RadioArticle:lowres:photo1.tif
%ALDImageID: OPIServer:Disk1:941106:CarlsenJob:hires:photo1.tif
%ALDImageDimensions: 16 16
%ALDImageCropRect: 0 0 16 16
%ALDImagePosition: 144 648 144 720 216 720 216 648
%%BeginObject: image
%%BeginOPI: 2.0        % must immediately follow %%BeginObject:
                       % in a combined OPI 1.3/2.0 context
%%ImageFileName:(Server:Disk1:941106:CarlsenJob:hires:photo1.tif)
%%ImageInks:  monochrome 1  (Black) 1.0
                       % map unit square to the correct position
%%BeginIncludedImage
%%IncludedImageDimensions:  16  16
%%IncludedImageQuality:  1.0 16 16 8  %width  height  bits/sample
/_bits exch def
/_h exch def
/_w exch def
/imbuf _w _bits mul 7 add 8 idiv string def
_w _h _bits [_w 0 0 _h neg 0 _h] {currentfile imbuf readhexstring
pop}
%%BeginData: 539 Hex Bytes
```

```
OPIimage
000102030405060708090A0B0C0D0E0F
10111213141516171819191A1B1C1D1E1F
202122232425262728292A2B2C2D2E2F
303132333435363738393A3B3C3D3E3F
404142434445464748494A4B4C4D4E4F
505152535455565758595A5B5C5D5E5F
606162636465666768696A6B6C6D6E6F
707172737475767778797A7B7C7D7E7F
808182838485868788898A8B8C8D8E8F
909192939495969798999A9B9C9D9E9F
A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF
D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF
F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
%%EndData
%%EndIncludedImage
%%EndOPI
%%EndObject
...restore...
```

6 Design and Usage

6.1 Design Requirements

OPI 2.0 was designed with the following requirements in mind:

- The design must not be tied to particular applications or operating systems.

- The design must not require a special PostScript interpreter in order to perform high resolution image substitution. It must not require a PostScript interpreter at all. An ordinary application program must be able to find the necessary information and do the right thing.

- The design must carry enough information so that an OPI Consumer can alert users to problems such as missing high resolution images, and let them re-link to the correct image.

- The design must support both desktop prepress software and non-desktop prepress systems.

- The design must be as easy as possible for an OPI Consumer to implement.

- The design must support composite printing as well as pre-separated printing.

- The design must be compatible with Adobe Document Structuring Conventions.

- The design must support LanguageLevel 1 as well as LanguageLevel 2 and 3 printers, imagesetters, and platesetters.

- The design must support black and white, grayscale, and color images in any color space that is supported by the PostScript language. Images can either be pre-separated into CMYK or other ink-based space, or they can be represented in a device-independent color space and separated later in the process.

6.2   Workflow Options

Option 1

The standard way of using OPI is for a user to make a high resolution scan of an image (stored as a TIFF file, Scitex CT image, PDF file, or any other convenient format) on a photo editing workstation, and then make a low- or medium-resolution version of the high-resolution image. The high-resolution image is archived on an image substitution server. The low-resolution image is transmitted to the person doing the page layout, where it is placed on a page in the layout application. When the publication is finished, a PostScript language file is created by the application, with OPI comments in place of image data. The end user takes this PostScript language or PDF file to the image substitution server workstation for high-resolution image substitution. Output can be sent directly to OPI-aware spoolers, without explicitly creating a PostScript or PDF file.

Option 2

Another way of using OPI is to create the low-resolution proxy image as an EPS file with OPI comments pointing to the high-resolution image at the image substitution server, and send the resulting EPS file to the page layout workstation. This has the advantage of not relying on the page layout program to create correct OPI comments.

- There are some disadvantages to taking this approach to picture replacement:

- Such a proxy EPS/OPI file often stores several versions of the image data in order to create the screen preview and make proof printing to black-and-white and color printers work. Therefore, the EPS file will typically be considerably larger and print more slowly than an equivalent TIFF file.

- Cross-platform transfer may be a problem due to platform-specific methods of storing EPS screen previews.

- Page layout applications may not be able to assign a color to a black-and-white or grayscale image if it is in EPS form.

- Cropping a graphic in this form does not throw away the cropped data, so that more unused image data is sent to the printer, resulting in slower printing.

7 Reading from a named file

In some environments it may be useful to have the PostScript RIP read the data for a high-resolution image directly from disk, using the PostScript operator file to set up a file object.

Note The file must contain only image data, and not any executable PostScript language code. In a LanguageLevel 2 or 3 environment, the data can be compressed, ready to be decompressed using one of the PostScript language decompression filters.

The only obstacle to doing this is that a page layout program may have cropped the image, requiring corresponding cropping of the high-resolution image in order to achieve the correct cropping, scaling, and aspect ratio.

There are at least two ways to overcome this obstacle. The preferred method is for the Consumer to pre-scan the PostScript language file containing the OPI comments, and use the OPI information to create a cropped version of the high resolution image data before proceeding to the OPI substitution phase.

The other option for the Consumer is to use the entire image, without cropping it, but scaling the coordinate system so that the uncropped part of the image maps onto the unit square, while clipping to the unit square. This method is easy to implement, but has the drawback that more image data than necessary is processed by the imagesetter.

C code to implement the second option might look something like this:

```
/* (x0,y0) and (x1,y1) are opposite corners of the croprect */
A = ncols / (x1-x0);
D = nrows / (y1-y0);
E = -x0 / (x1-x0);
F = -y0 / (y1-y0);
WritePS ("[%f 0 0 %f %f %f] concat\n", A,D,E,F);
```

Warning: an OPI Consumer must not read directly from a named disk file unless it is sure that it is the either the last program in the prepress pipeline, or that at least the file system known to the RIP is static. Otherwise the RIP will not be able to find the image data.

8 Change History

The following section details changes made to the OPI specification since version 1.3.

Changes from Version 1.3 to Version 2.0

- The work of an OPI Consumer has been greatly simplified, due to new conventions to isolate the setting of image characteristics from the image data itself. To facilitate this transition, new %% OPI comments have been introduced. The old %ALD comments are typically ignored by an OPI 2.0 Consumer if OPI 2.0 comments are present. %ALD comments are no longer described in this document.

- The examples have been updated to use the new comments.

- A number of sections have been clarified, rewritten, or reorganized.

- Guidelines for supporting EPS proxy and high resolution images and PDF main image were added.

- The syntax for filenames and ASCII fields was clarified and made consistent with DSC syntax.

- The document was reformatted to be visually compatible with Appendix G: Document Structuring Conventions—Version 3.0, in the *PostScript Language Reference Manual, Second Edition.*