Adobe Developers Association

ADA news

In This Issue

Volume 3, Number 8

An Overview of the CID-keyed Font Format for Far Eastern Fonts

If you are a font developer who publishes fonts for Far East markets (in the Chinese, Japanese, or Korean languages), you should be aware of Adobe's CID-keyed font format and its benefits for you. Also, if you develop application software which supports Far East languages, this format and its implications should be of interest.

The CID-keyed font file format was designed for large character set fonts for use with PostScript™ language printing, Adobe Type Manager™ (ATM™) software, Configurable PostScript Interpreter (CPSI) software, and Display PostScript™ (DPS) software. It is the ideal format for Chinese, Japanese, or Korean (CJK) fonts, and may also be used for Roman fonts with very large character sets. In fact, the technology is independent of Far East languages, and could be used with any writing system.

The CID-keyed font file format is an open, published format which is specified in Adobe Technical Specification #5014, "The CMap and CID-Keyed Font Files Specification." This document specifies a new *file organization* that allows optimal flexibility and performance. The characters contained in a CID-keyed font are in the standard PostScript Type 1 format that is the industry standard for high-quality, cross-platform printing.

A key advantage of the CID-keyed font file format is the ease with which font developers can support a wide variety of operating systems, character sets, and encodings. It makes it easier for vendors to offer better products, and users benefit from the format's compatibility with a wide range of PostScript output devices. Performance will be considerably enhanced with future PostScript printers and versions of Adobe Type Manager that will directly interpret CID-keyed fonts. Further performance enhancements will be seen with the new PostScript products for the Asian market that have hardware-assisted font rendering capabilities.

Overview of the CID-Keyed Font Architecture

Previously, Adobe™ used the Original Composite Font (OCF) format for fonts for use with ATM-J software and Japanese printer products. While OCF format fonts offer very high quality and will continue to be used by many users, the format does not offer the flexibility and simplicity that are key benefits of the CID-keyed format. The two formats will remain completely compatible, allowing users to freely use both kinds of fonts in their systems.

Figure 1 illustrates the general structure of an OCF and a CID-keyed font as they appear in the memory of a PostScript interpreter. An OCF font program sets up a complex structure of intermediate "fonts" to be able to access all of the characters in accordance with the various encodings supported by a single font. CID-keyed fonts set up a much simpler structure, as shown in the diagram. Because of the simpler structure, CID-keyed fonts use less memory, and the interpreter can retrieve and rasterize character outlines much faster.

continued on page 2

Adobe™

How To Reach Us

DEVELOPERS ASSOCIATION HOTLINE:

U.S. and Canada:
(415) 961-4111
M–F, 8 a.m.–5 p.m., PDT.
If all engineers are unavailable, please leave a detailed message with your developer number, name, and telephone number, and we will get back to you within 24 hours.

Europe:
+31-20-6511-355

FAX:

U.S. and Canada:
(415) 967-9231
Attention:
Adobe Developers Association

Europe:
+31-20-6511-313
Attention:
Adobe Developers Association

EMAIL:

U.S.
devsup-person@mv.us.adobe.com

Europe:
eurosupport@adobe.com

MAIL:

U.S. and Canada:
Adobe Developers Association
Adobe Systems Incorporated
1585 Charleston Road
P.O. Box 7900
Mt. View, CA 94039-7900

Europe:
Adobe Developers Association
Adobe Systems Europe B.V.
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z.O.
The Netherlands

Send all inquiries, letters and address changes to the appropriate address above.

CID-keyed Font Format

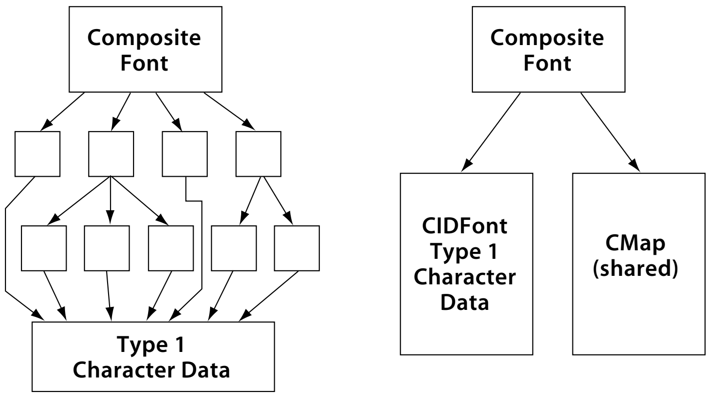

Figure 1. *The structure of OCF and CID-keyed fonts in memory.*

Figure 2 illustrates both the sequence of development and the relationship between the components of a CID-keyed font.

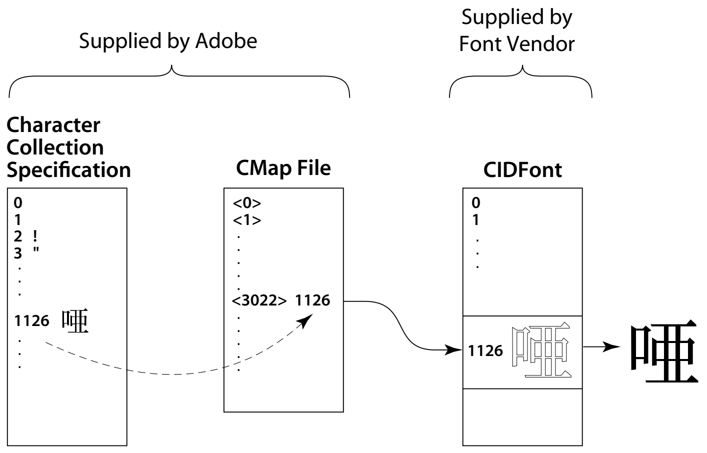

Figure 2. *Sequence and relationship of files used for CID-keyed fonts.*

The sequence of development is as follows:

- Adobe Systems develops a *character collection* document (see following section on character collections) for a specific language. This document specifies all of the characters (and their CID numbers) needed for that language.

CID-keyed Font Format

• Adobe Systems develops one or more *CMap files* for that character collection (described below). These files specify the correspondence between character codes and CID numbers for popular character sets and encodings.

• Other font developers may easily define their own character collections and CMap files, but it is expected that the ones supplied by Adobe will fit the needs of most font vendors.

• Font developers then only need to produce their *CIDFont* files, which contain the character shapes. These fonts reference the CMap files and character collection definitions created by Adobe.

This approach minimizes the amount of work a font developer has to do, yet still allows the necessary flexibility for them to use any non-standard character set or encoding that they wish.

CID-Keyed Font Components

The name *CID-keyed Font* refers to *Character ID* (CID) numbers that are used to index and access the characters in the font. This method is more efficient for large fonts than the method of accessing characters by character name, as is used for Type 1 Roman fonts.

A CID-keyed font consists of one or more *CMap files* and a *CIDFont file*. The Character ID numbers in the font are based on a predefined and named *character collection*, and a specific *ordering* of that collection. These components are discussed in the following sections.

Character Collections

A *character collection* consists of an ordered set of all characters needed to support one or more popular character sets for a particular language. The order of the characters in the character collection determines the CID number for each character, and is the basis for the correspondence between character codes and CID numbers referenced in the CMap file. Each CID-keyed font must explicitly reference the character collection upon which its CID numbers are based.

For example, the Adobe character collection for the Chinese simplified character set is *Adobe-GB1-0*, where *Adobe* is the registry name; *GB1* represents the first version of the ordering of the Chinese GB 2312-80 character set, and 0 (zero) specifies that it is the base collection, to which additional supplements may be added. This naming convention allows for strict version and compatibility control, while also allowing the flexibility required by font developers.

Adobe Systems publishes named character collections for Chinese, Japanese, and Korean fonts; other font developers can reference these character collections, or they can develop and name their own character collections.

continued on page 4

CID-keyed Font Format

CMap Files

CMap (Character Map) files specify the correspondence between a character code and the CID number used to access the character description in the CIDFont file. The CMap file is equivalent to the concept of an *encoding* as used in the Type 1 font format. Whereas a Type 1 font allows a maximum of 256 characters to be encoded and accessible at one time, users can access all of the thousands of characters in a very large CID-keyed font.

Figure 3 illustrates a CMap file and CIDFont file, and how character codes and character IDs are used to access characters in a CID-keyed font.

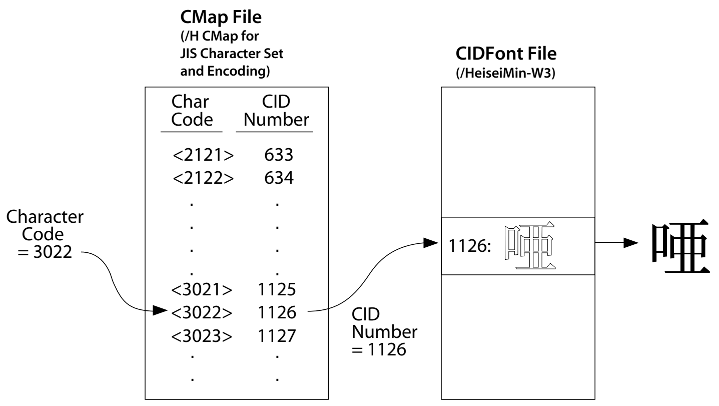

Figure 3*. Accessing characters with CMap and CIDFont files.*

A CMap file can reference an entire character collection, or only a subset. It can also reference any other CMap file, without having to duplicate it, to provide extra font configurations (see section entitled, “Rearranged Fonts”).

An increasingly common question asked of most font technologies is whether they can support the ISO 10646 and Unicode® character encoding standards. The CID format does provide for the double-byte character addressing that is required for Unicode fonts. The only other requirement is the addition of a CMap file, which Adobe provides, that specifies the correspondence between Unicode character code values and the CID numbers contained in the font. This inherent flexibility makes it very easy for developers to add Unicode support to their font products.

CIDFont Files

The CIDFont file contains the characters in the font, each of which is a computer language procedure that “draws” a given character shape for either display on the screen or for printing. The CIDFont file is the only component that most font developers will have to produce. The

CID-keyed Font Format

character procedures are in the same format as those in a Type 1 font, which is why CID-keyed fonts can be made compatible with many PostScript interpreters by adding, to the font package, a compatibility module that is installed in the user’s printer.

The CIDFont file also contains tables that help the interpreter locate the required characters and associated data, as well as additional information called *hints* which help the interpreter to create high-quality images of the characters at small sizes or at low resolutions.

New Counter Control Hints

One type of hint information that is particularly important for complex CJK characters is the *Counter Control* hint (*counters* are the white spaces between strokes in a character). It helps ensure that counter spaces and overall proportions are rendered as accurately as possible, subject to the number of pixels available for a given size and resolution. This specialized type of hint information is not needed for ordinary Type 1 Roman fonts, but adds an essential degree of control for complex characters such as the one shown in Figure 4. Failure to add these hints can significantly hurt the performance and quality of a font when used with PostScript printers with the Type 1 Coprocessor.

Figure 4a shows the Chinese character that represents *rumble* (the sound), and b and c show the bitmap which is created for display at 24-point on a 72 dots-per-inch screen. The bitmap character in b shows the effect of having Counter Control hints in the font, and c shows how one of the counter spaces can collapse if these hints are not included.

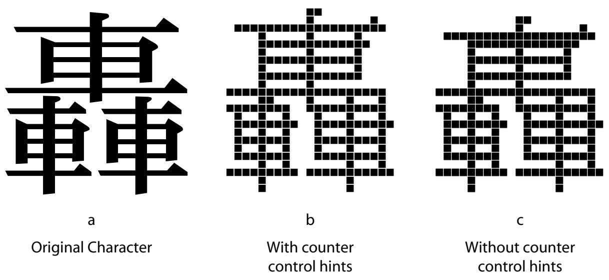

Figure 4. *Example of the results of rendering a character with, and without, Counter Control hints.*

Rearranged Fonts

One of the most powerful features of the CID-keyed font file format is the ability to create a *rearranged* font. This feature allows font developers to create a font which consists of only a CMap file that references other fonts installed in the user’s system. It may reference one or more CID-keyed, OCF, Type 1, or Type 3 fonts. This feature provides great flexibility with minimal development effort and file storage requirements.

continued on page 6

Don't Forget *to register!*

If you have purchased a Software Development Kit (SDK), be sure to return the registration card. We will be incorporating new features into the SDKs in the future, and want to be sure to inform you about new additions.

CID-keyed Font Format

A rearranged font is created using a CMap file. There are no actual characters in the rearranged font; the CMap file serves only as a template which describes the fonts from which characters are to be borrowed. It also contains a specification of how those characters correspond to the input codes. The rearranged font behaves the same way as any other CID-keyed font—its name appears in font menus, it can be downloaded to a printer, and it can be used with ATM software. The advantage is that the size of the resulting CMap file is typically less than 30 kilobytes. Although the referenced font files must be in the user's system, this approach avoids the need for duplication of any of the component fonts.

Figure 5 illustrates an example of a rearranged font which "borrows" characters from two CID-keyed fonts, references two Type 1 fonts for alternate Roman and symbol characters, and one Type 3 font that may contain one or more company logos (Type 3 fonts are useful for complex artwork that may be unsuitable as a Type 1 font).

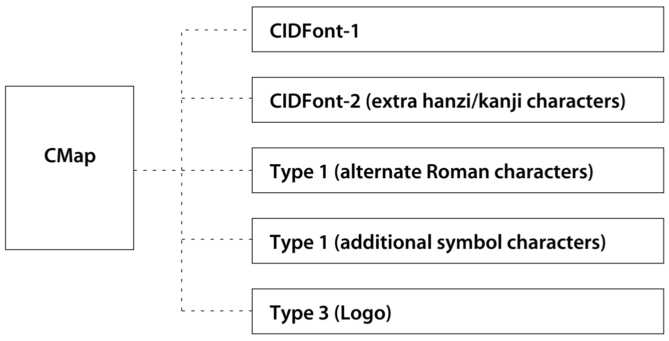
CMap
CIDFont-1
CIDFont-2 (extra hanzi/kanji characters)
Type 1 (alternate Roman characters)
Type 1 (additional symbol characters)
Type 3 (Logo)

Figure 5. *CMap file used to create a rearranged font.*

Benefits of the CID-Keyed Font Format

Although benefits for end users and for font developers are listed separately in the following paragraphs, the two are closely related. What makes users satisfied is good for the font vendor, and what helps vendors make better fonts makes it possible for them to offer better products to satisfy their users.

User Benefits

User benefits can be grouped into two categories: performance and compatibility. For newer PostScript products (PostScript version 2015 and later), CID fonts will be interpreted in *native mode*, which means the user will experience optimal performance and ease-of-use when using CID-keyed fonts with such PostScript printers.

CID-keyed Font Format

Performance

The performance of CID-keyed fonts is comparable to OCF fonts with existing printers, but performance will be up to 50 percent faster with future native-mode printers (though performance is also dependent on the quality of the PostScript language code created by the user's printer driver).

Complex CJK fonts can create a performance problem because of the large number of characters that must be imaged for an average print job. The Adobe™ Type 1 Coprocessor, an ASIC, significantly enhances the performance of CID-keyed fonts. The recent development of an inexpensive version of this coprocessor will ensure its wide availability in a variety of new printers for the Asian market.

Table 1 shows the rendering speeds, at 12-point and at 300-dpi, of three kinds of characters, with the Type 1 Coprocessor and with a RISC-based font interpreter. These numbers include the required overhead time for parsing and decrypting the font program, and for rasterizing, caching, and transferring the characters to the frame buffer for printing.

	RISC-based Controller *(characters-per-second)*	**Type 1 Coprocessor** *(characters-per-second)*	**Percent Increase**
Roman	94	370	293%
Kanji:			
OCF	38	175	360%
CID	41	250	509%

Table 1. *Rendering speeds for Type 1 Coprocessor and RISC Controller*

Compatibility

Because the characters in a CID-keyed font are in the standard Type 1 font format, they are compatible with a wide variety of operating systems, applications software, and output devices. For comparison, TrueType™ fonts must be translated into Type 1 fonts to be interpreted by PostScript imagesetters, and many service bureaus only accept jobs which utilize Type 1 fonts. Also, ATM software enables users to use CID-keyed fonts when printing to a wide variety of non-PostScript printers.

continued on page 8

CID-keyed Font Format

Figure 6 illustrates the versatility of CID fonts by showing some of the output device choices available.

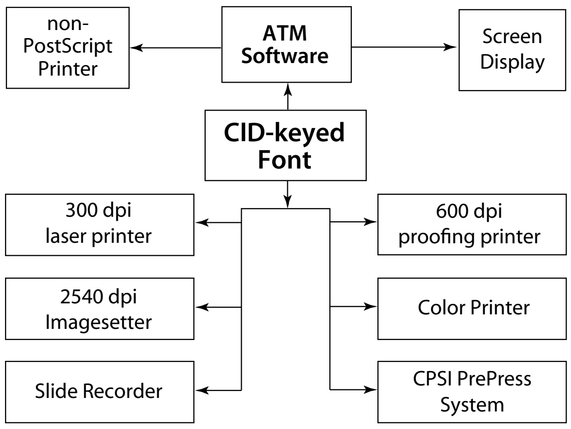

Figure 6. *Compatibility of CID-keyed fonts with a variety of output devices.*

In addition to the performance and compatibility benefits explained above, end users benefit from the simplicity and smaller size of CID-keyed fonts which makes them easier to install and maintain.

Benefits for Font Developers
While users tend to focus on the quality and performance benefits of a font, it is equally important to realize the benefits offered to font developers. It can have a major effect on what is available to users in terms of the variety and quality of font products. All font developers are concerned about production efficiency, flexibility of the resulting products, and the need to protect their typeface designs; but these issues are particularly important to developers producing CJK fonts because of the large investment necessary to design, produce, and test those fonts.

The CID font format offers font developers the following advantages:

• ***Simplicity:*** Reduces the number of files needed from nearly 100 for OCF fonts to only one for CID-keyed fonts (CMap files, while required to make use of the CID-keyed font, are shared among all CID-keyed fonts). The simplicity of the format results in approximately a 15 to 20 percent savings in file size (see Table 2), and thus the font uses less of the printer's memory. This simplicity also means that fonts are easier to build, install, and verify.

CID-keyed Font Format

	OCF	CID	CID: % smaller
Ryumin-Light™	5.463 MB	4.618 MB	15%
Gothic Medium BBB™	4.125 MB	3.223 MB	20%

Table 2. *Sample file sizes for OCF and CID fonts (file sizes in megabytes)*

Note: The CID versions of these fonts evaluated for this example contain approximately 150 more characters than their OCF counterparts.

• ***Extensible design:*** Support for additional character sets and encodings may be easily added to existing fonts. Font developers may start with support for a limited character set and then add characters, in a modular fashion, to reach additional markets. This is probably the single-most valuable feature of the CID-keyed font format for most developers.

• ***Quality:*** The new format continues to allow the addition of standard Type 1 hint information, which greatly improves the quality for small point sizes and low resolutions (see Figure 4 for an example of the effect of hint information on rendering quality).

• ***Portability:*** A single font file will work with ATM software (version 3.8), PostScript printers or imagesetters, CPSI software, and Display PostScript applications for the Macintosh®, Microsoft® Windows™, and UNIX® environments. This means a significant savings in development and packaging costs for the font developer, and enhanced ease-of-use for the end user.

• ***Elimination of redundancy:*** Once a CMap file is in memory, it can be shared across all fonts, whether they are full or subset fonts.

• ***Font Protection:*** CID-keyed fonts are computer programs, and therefore may be copyrighted like any software product. However, developers of large CJK language fonts often want some form of physical protection since each font is a much larger investment than that needed for a Roman-alphabet font. Physical copy protection mechanisms are as easy with CID-keyed fonts as with OCF fonts.

• ***Support:*** The OCF format is Adobe proprietary, while the CID format is published and fully supported by Adobe Systems.

In summary, these features offer developers the best potential to provide high-quality fonts with the kind of flexibility needed to meet market demands.

continued on page 10

CID-keyed Font Format

Compatibility

The CID-keyed font format was designed for maximum compatibility with a wide range of system and applications software, as well as with most PostScript interpreters. Adobe Systems tests CID-keyed fonts with a wide variety of applications that support CJK language fonts, and actively works to resolve any incompatibilities which are found.

CID-keyed fonts will be interpreted directly, in native mode, by version 2015 or later of the PostScript interpreter. For printing with pre-2015 versions, a *CID Support Library* module (available from the Adobe Developers Association) must be included in the font. This module makes CID-keyed fonts backwards compatible with existing PostScript Level 2 interpreters (version 2011 or later), and with Level 1 PostScript Japanese printers.

CID-keyed fonts are compatible with most software applications. In order for the applications to access the character metrics (widths), CID-keyed fonts must have the font metrics file appropriate for the operating environment. This includes a suitcase file for the Macintosh, a PFM (Printer Font Metrics) file for Windows, or an AFM (Adobe Font Metrics) file for UNIX systems.

The format is also compatible with ATM 3.8 for the Macintosh, and with ATM for Windows 3.0J. ATM software, version 3.8 for the Macintosh, will also allow the use of CID-keyed fonts with the Apple® QuickDraw™ GX operating system that will ship with System 7.5.

Recommendations for developers

The CID-keyed font file format offers major advantages over the OCF format. Font developers should definitely choose to develop CID-keyed fonts for the best quality, performance, and compatibility with PostScript printing. It is also essential for them to put Counter Control hint information in their fonts to achieve the highest quality and performance. Information and materials to assist in the development of CID-keyed fonts are available from the Adobe Developers Association.

If you are an application developer, you should not need to know about CID fonts or how they are structured; they should work transparently with your application on the host computer. If your application or driver does depend on the structure of either OCF or CID-keyed fonts, this can be somewhat dangerous in terms of forward compatibility. In future articles, we plan to discuss how you can either avoid dealing with the structure of the font, or do it in a safe manner. If you need to solve this problem in the very near future, please contact the ADA for help in this matter. §

Questions Answers

In the Questions and Answers section of the *ADA News* (volume 3, number 4), we recommended putting Level 2 style dictionaries into strings, to ensure portability on Level 1 devices. The code example we gave was as follows:

```
mark
{ /testdict (<</First (John) /Last (Smith)>>) cvx exec def
} stopped cleartomark
```

The intent of the Q&A was to show how to avoid syntax errors that can't be trapped by the **stopped** operator when executing the code with a Level 1 interpreter. In response to this Q&A, Alan Macdonald, a member of our ADA in Europe, recommended an alternative method which requires less memory for the string. The method he has used to overcome the same problem is to put only the level 2 syntax into the strings. So, the code would be as follows:

```
(<<) cvx exec /key value ... (>>) cvx exec
```

In addition to the memory savings, another advantage to this method is that you are not limited to the string size limit of 64 Kbytes for your dictionary definition.

If your PostScript language output defines procedure sets in its prolog, then you could incorporate Alan's suggestion into your procedure sets as shown in the following code sample:

```
%!PS-Adobe-3.0
%%Title: (Code sample with Level 2 dict constructs)
%%EndComments

%%BeginProlog
/L2? {
      /languagelevel where {
         pop languagelevel
         2 eq {
           true
         }{
           false
         } ifelse
     }{
         false
     } ifelse
} def
```

```
% define dictionary constructors _dd and dd_ (like << and >>)
L2? {
      /_dd (<<) load def
      /dd_ (>>) load def
}{
      /_dd mark def
      /dd_ {
            counttomark 2 idiv dup dict begin
            {def} repeat
            pop                 %remove mark from stack
            currentdict end
     } bind def
} ifelse

/Mydict _dd
    /m /moveto load
    /l /lineto load
    /f /fill load
    /s /stroke load
dd_  def
%%EndProlog

%%Page: 1 1
Mydict begin
100 100 m 200 100 l 200 200 l closepath
gsave 0.5 setgray f
grestore s
end
showpage
%%EOF
```

Using the above definitions in your procset would be useful if you want to build a dictionary on-the-fly in your PostScript language output, without knowing how many entries the dictionary will contain. If you are willing to pay an execution time penalty on Level 1 devices, this code allows you to create dictionaries, without concern for whether your code is executing in a Level 1 or Level 2 environment. Do note that dictionaries created with this method on Level 1 devices still cannot be expanded the way Level 2 dictionaries can. Also note that in the code above, we use (<<) `load` rather than (<<) `cvx`. Either operator works, but using **cvx** will cause the name lookup to be done every time _dd or dd_ is called, while using **load** causes the lookup to be done just the one time they are defined. Assuming these procedures are going to be used a multiple number of times, early binding of the names << and >> is preferable.

Q Our application saves the information returned by currentscreen, and later restores these values with a call to setscreen. The code worked fine under Level 1 PostScript, but we are now having problems on Level 2 PostScript printers. What is wrong with our code?

The code used is as follows:

```
currentscreen
/spotfcn exch def
/angle exch def
/freq exch def

....

freq angle {spotfcn} setscreen
```

On a level 2 device, this code yields the following error:

```
%%[ Error: typecheck; OffendingCommand: setscreen ]%%
```

A This behavior of **currentscreen** and **setscreen** is addressed in Technical Note #5119 "Level 2 Compatibility: The setscreen and currentscreen Operators." Under Level 1 of the PostScript language, the **currentscreen** operator returned the frequency and angle of the screen as well as a procedure describing the spot function, as follows:

— **currentscreen** frequency angle proc

Under Level 2, we can now use **sethalftone** to set the screens on a printer. If the screen is set in this manner, **currentscreen** will return parameters of the following type:

— **currentscreen** 60 0 halftonedict

or if a Type 1 halftone dictionary was specified:

— **currentscreen** freq angle halftonedict

Similarly, the **setscreen** operator can be called with two numbers and a halftone dictionary. For a list of other changes to existing operators, see section A.2 of the *PostScript Language Reference Manual, Second Edition*. The description of **setscreen** and **currentscreen** have been modified since the early printings of the *PostScript Language Reference Manual, Second Edition*. In addition to Technical Note #5119, also see Technical Note #5085 "Updates to the PostScript Language Reference Manual, Second Edition."

To return to our example code, we need to write it so that it will work regardless of whether **currentscreen** returns a procedure or a dictionary on top of the stack. Replacing "`freq angle {spotfcn} setscreen`" with "`freq angle /spotfcn load setscreen`" will accomplish this.

On a final note, if your application is trying to control the printer's screening via the **setscreen** operator, you may want to consider why you are doing this. For many printers, calling **setscreen** is not at all appropriate.

Q Why do you say calling setscreen is not always appropriate?

A If your application calls **setscreen**, you are making two assumptions: that the output device does halftoning, and that the halftoning can be controlled in a traditional manner by **setscreen**.

Some printers can print continuous tone colors. Therefore, they don't do any halftoning at all. On these printers, **setscreen** generally has no effect.

Other printers use halftoning which is different from conventional screening. For instance, Adobe Brilliant™ Screens implement Frequency Modulation (FM) screening. With FM Screening the halftone consists of constant-size dots, in a seemingly random pattern that becomes more dense as the color intensity increases. Brilliant Screens should start appearing on imagesetters from Monotype and other Adobe OEMs in the near future. Another type of non-conventional screening exists on Apple's LaserWriter® IIf and IIg printers. These devices implement a feature called PhotoGrade,™ which uses the laser-printing engine to improve the quality of the halftoning. Such non-conventional halftoning may not support a traditional concept of frequency or angle.

Think about why you are calling **setscreen** in the first place. If you are trying to give users control over halftoning, realize that the device's own default halftoning may be better than

continued on page 13

Questions & Answers

anything you can control by **setscreen**. Your application should always let the user disable the call to **setscreen** and rely on the default halftoning. Or, if your application's intent is to implement special effects, such as line screening, you may need a different strategy for devices with continuous-tone output or non-conventional halftoning.

In summary, if your application calls **setscreen**, it should be a warning that you may be making incorrect assumptions about the kind of halftoning the user's output device performs. Check your assumptions carefully, and always provide the user a way to turn off the call to **setscreen**. §

Colophon

This newsletter was produced entirely with Adobe PostScript software on Macintosh and IBM® PC compatible computers. Artwork was produced using Adobe Illustrator™ 5.0 and Quark XPress® software. Typefaces used are from the Minion™ and Myriad™ families, all of which are from the Adobe Type Library.

Managing Editors:
Nicole Frees, Debi Hamrick

Technical Editor:
Jim DeLaHunt

Art Director:
Gail Blumberg

Designer:
Karla Wong

Contributors:
Tim Bienz, Jim DeLaHunt, Nicole Frees, Terry O'Donnell

Adobe, the Adobe logo, Acrobat, Adobe Brilliant, Adobe Illustrator, Adobe Type Manager, ATM, Display PostScript, Distiller, Minion, Myriad, PostScript, and the PostScript logo are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. Apple, LaserWriter, and Macintosh are registered trademarks and PhotoGrade, QuickDraw and TrueType are trademarks of Apple Computer, Inc. IBM is a registered trademark of International Business Machines Corporation. Quark Xpress is a registered trademark of Quark, Inc. Ryumin-Light and Gothic Medium BBB are trademarks of Morisawa & Co., Ltd. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. IBM is a registered trademark of International Business Machines Corporation. All other brand and product names are trademarks or registered trademarks of their respective holders.
©1994 Adobe Systems Incorporated.
All rights reserved.

Part Number ADA0048 8/94

Adobe PostScript™

ACROBAT column™

The PDF file format, and the Adobe Acrobat™ software that supports it, provide a wide variety of data compression methods to minimize the size of PDF files, while maintaining the visual integrity of documents— LZW, Run Length, CCITT Group 3 and 4, and JPEG compression methods are supported. The setdistillerparams operator, which you place in your application's PostScript language output, allows you to control many of the compression options used when a file is converted to PDF by the Adobe Acrobat Distiller™ program. This column describes several of the many reasons why you should consider using this operator in your program's PostScript language output.

Although many of the compression settings are available through the Distiller program's user interface, there are three reasons to control them using `setdistillerparams`. First, by embedding the settings in the PostScript language file, you ensure that the same results are obtained each time the document is distilled, regardless of how a user might have changed the compression settings in the user interface. Second, many users rely on a Network Distiller running on a machine to which they do not have physical access, and so they are unable to use the Distiller program's user interface to change the compression settings. Finally, there are times where different compression settings are appropriate for different parts within a single document— `setdistillerparams` is the only way to provide this ability.

Documents containing both digitized photographs and grayscale or color screen shots are examples of documents that benefit greatly from image-by-image compression control. Digitized photographs generally compress extremely well, with little loss of quality, using JPEG compression, but screen shots must be LZW compressed to retain the sharp lines they usually contain. Using LZW compression for all images in such a document usually causes the PDF file size to be substantially larger than it could be, while using JPEG compression for all images generally results in objectionable visual degradation of the screen shots.

There are many other reasons to support `setdistillerparams`. For further information, see Technical Note #5151 “Acrobat Distiller Parameters” (in the Adobe Acrobat Developer Compatibility Test Kit). Also see Technical Note #5150, “pdfmark Reference Manual” for a description of another special operator that the Distiller program supports. §