Building PFM Files for PostScript-Language CJK Fonts

Adobe Developer Support

Technical Note #5178

10 January 1997

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110
(408) 536-6000 Main Number
(408) 536-9000 Developer Support
Fax: (408) 536-6883

European Engineering Support Group
Adobe Systems Benelux B.V.
P.O. Box 22750
1100 DG Amsterdam
The Netherlands
+31-20-6511 355
Fax: +31-20-6511 313

Adobe Systems Eastern Region
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120
Fax: (617) 273-2336

Adobe Systems Co., Ltd.
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150
Japan
+81-3-5423-8169
Fax: +81-3-5423-8204

PN LPS5178

Copyright © 1996 – 1997 by Adobe Systems Incorporated. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher. Any software referred to herein is furnished under license and may only be used or copied in accordance with the terms of such license.

PostScript is a trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Any references to a "PostScript printer," a "PostScript file," or a "PostScript driver" refer to printers, files, and driver programs (respectively) which are written in or support the PostScript language. The sentences in this book that use "PostScript language" as an adjective phrase are so constructed to reinforce that the name refers to the standard language definition as set forth by Adobe Systems Incorporated.

Adobe, Adobe Type Manager, ATM, Display PostScript, PostScript and the PostScript logo are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. Microsoft and Windows are registered trademarks of Microsoft Corporation. CJK is a registered trademark and service mark of The Research Libraries Group, Inc. All other brand or product names are the trademarks or registered trademarks of their respective holders.

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 warranty 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 noninfringement of third party rights.

Contents

Building PFM Files for PostScript-Language CJK Fonts

1 Introduction

Printer Font Metrics (PFM) files are required for installing Type 1, CID-keyed, or composite font programs in a Windows® system. Although this file format was originally designed for single-byte fonts, PFM files provide Adobe Type Manager® (ATM®) and other software with key font-specific information that is necessary for installation and use in a Windows environment.

There are some special considerations when building PFM files for PostScript®-language CJK® (Chinese, Japanese, and Korean) fonts. This Technical Note describes the structure of a PFM file in practical terms, and supplies guidance that can be used for building tools that generate well-formed PFM files. A simple tool written in Perl for building PFM files is included as part of this document (see Appendix A). The PFM file specification is available in the Microsoft Windows Device Development Kit (DDK), available from Microsoft. That document describes the structure and use of PFM files only for single-byte fonts.

PFM files are packed as binary structures with many sections, and contain information about a single font. Many of the data fields in PFM files are two- or four-byte values that must be in little-endian byte order (that is, the byte order used on Windows systems). They represent a single numeric value, but are packed into data sizes greater than one byte. Such two-byte entities are often called “WORD” or “short” values, and four-byte entities are often called “DWORD” or “long” values.

Many of the font-specific values that are included in a PFM file, such as ascent and descent information, can be calcu-

lated or read from the font data itself. The availability of an AFM (Adobe Font Metrics) file can ease or trivialize these calculations. For CID-keyed fonts, the CID-keyed AFM file format is described in Adobe Technical Note #5004: "Adobe Font Metrics File Format Specification," Version 4.1.

2 PFM File Sections

There are five sections that comprise PFM files for PostScript-language CJK fonts, indicated as follows:

Header
Extension
Extended Text Metrics
PostScript Information
Extent Table

Values for most PFM fields do not change from one font to another — such values are static. And, some fields are not currently used by drivers or applications, but it is wise to calculate reasonable values in case they are used in the future.

3 PFM Header

The PFM file's *Header* section, 117 bytes total size, is described in Table 1.

Table 1 *Header*

Field Name	Data Size	Value
dfVersion	Two bytes	256
dfSize	Four bytes	The exact size of the PFM file, in bytes
dfCopyright	60 byte string, null padded	Copyright string
dfType	Two bytes	129
dfPoint	Two bytes	10
dfVertRes	Two bytes	300
dfHorizRes	Two bytes	300

Table 1 *Header (Continued)*

Field Name	Data Size	Value
dfAscent	Two bytes	Font-level ascent (the fourth element of a FontBBox array)
dfInternalLeading	Two bytes	Internal leading (calculation shown below)
dfExternalLead-ing	Two bytes	196
dfItalic	One byte	0
dfUnderline	One byte	0
dfStrikeOut	One byte	0
dfWeight	Two bytes	400 or 700 (calculation shown below)
dfCharSet	One byte	128, 129, 134, or 136 (calcula-tion shown below)
dfPixWidth	Two bytes	0
dfPixHeight	Two bytes	1000
dfPitchAndFamily	One byte	16, 17, 48, 49, 64, or 65 (calcula-tion shown below)
dfAvgWidth	Two bytes	500
dfMaxWidth	Two bytes	1000
dfFirstChar	One byte	32 (0x20)
dfLastChar	One byte	255 (0xFF)
dfDefaultChar	One byte	0
dfBreakChar	One byte	0
dfWidthBytes	Two bytes	0
dfDevice	Four bytes	199 (offset value to the string “PostScript” in the “PostScript Information” section, in bytes)
dfFace	Four bytes	210 (offset value to the Win-dows Name string in the “Post-Script Information” section, in bytes)
dfBitsPointer	Four bytes	0
dfBitsOffset	Four bytes	0

The *dfSize* value is calculated after all other field sizes are known. The only two variable-length PFM fields are in the "PostScript Information" section, and are null-terminated strings that represent the Windows font menu name and the fully-qualified PostScript font name (for a definition of these entries, see section 6).

Note 1 How the printer driver or other software deals with the Windows font menu name or the fully-qualified PostScript font name is beyond the scope of this document.

An example *dfCopyright* string, without the null padding, looks like the following, taken from an Adobe Systems' PFM file:

```
Copyright 1985-1997 Adobe Systems Inc.
```

If the *dfCopyright* string exceeds 60 bytes, it needs to be shortened or truncated.

Notes about the calculations:

- The *dfAscent* value is calculated by determining the highest point in the font. This value can be extracted from the fourth element of the *FontBBox* (font bounding box) array found in an AFM file.

- The *dfInternalLeading* value is calculated as follows:

  If the result of (ascent – descent) is less than 1000, then *dfInternalLeading* should be set to zero.

  Otherwise, *dfInternalLeading* should be set to the value of ((ascent – descent) – 1000).

  The ascent value is the same as the *dfAscent* field, and the descent value can be extracted from the second element of the *FontBBox* array.

- The *dfWeight* value is specified as follows:

  For bold fonts, use 700
  For all others, use 400

- The *dfCharSet* value is specified as follows (supported encodings in parentheses):

Japanese = 128 (Shift-JIS)
Korean = 129 (EUC-KR or Unified Hangul Code)
Simplified Chinese = 134 (EUC-CN or GBK)
Traditional Chinese = 136 (Big Five)

This value tells Windows how to treat this font in terms of language, character set, and encoding.

- The *dfPitchAndFamily* value is specified as follows:

  Serif-like designs (Mincho, Song, Myungjo) = 16
  Script-like designs (Kaisho, Gyosho, Kai) = 64
  Other designs (including sans serif: Gothic or Hei) = 48

  (Note that the Japanese typeface design called *Kyokasho* is treated like a serif-like design.) Furthermore, if the pitch of the one-byte Roman characters is proportional, then add 1 to the above values.

4 PFM Extension

This section, 30 bytes total size, contains three meaningful fields, all of which are offset values to other structures of the PFM file. The format is shown in Table 2.

Table 2 *Extension*

Field Name	Data Size	Value
dfSizeFields	Two bytes	30 (the size of this section, in bytes)
dfExtMetricsOffset	Four bytes	147 (offset value to the “Extended Text Metrics” section, in bytes)
dfExtentTable	Four bytes	Offset value to the Extent Table, in bytes
dfOriginTable	Four bytes	0
dfPairKernTable	Four bytes	0
dfTrackKernTable	Four bytes	0
dfDriverInfo	Four bytes	Offset value to the fully-qualified PostScript font name string in the “PostScript Information” section, in bytes
dfReserved	Four bytes	0

5 PFM Extended Text Metrics

This section, shown in Table 3, includes many fields, and is 52 bytes in size.

Table 3 *Extended Text Metrics*

Field Name	Data Size	Value
etmSize	Two bytes	52 (the size of this section, in bytes)
etmPointSize	Two bytes	240 (12-point expressed in units of 1/20th of a point)
etmOrientation	Two bytes	0
etmMasterHeight	Two bytes	1000
etmMinScale	Two bytes	3
etmMaxScale	Two bytes	1000
etmMasterUnits	Two bytes	1000
etmCapHeight	Two bytes	Height of uppercase “H”
etmXHeight	Two bytes	Height of lowercase “x”
etmLowerCaseAscent	Two bytes	Height of lowercase “d”
etmLowerCaseDescent	Two bytes	Descent of lowercase “p” (absolute value)
etmSlant	Two bytes	0
etmSuperScript	Two bytes	–500
etmSubScript	Two bytes	250
etmSuperScriptSize	Two bytes	500
etmSubScriptSize	Two bytes	500
etmUnderlineOffset	Two bytes	100 (or 0 if vertical)
etmUnderlineWidth	Two bytes	50 (or 0 if vertical)
etmDoubleUpperUnder-lineOffset	Two bytes	50 (or 0 if vertical)
etmDoubleLowerUnder-lineOffset	Two bytes	100 (or 0 if vertical)
etmDoubleUpperUnder-lineWidth	Two bytes	25 (or 0 if vertical)
etmDoubleLowerUnder-lineWidth	Two bytes	25 (or 0 if vertical)

Table 3 *Extended Text Metrics (Continued)*

Field Name	Data Size	Value
etmStrikeOutOffset	Two bytes	405
etmStrikeOutWidth	Two bytes	50
etmKernPairs	Two bytes	0
etmKernTracks	Two bytes	0

Most of Adobe Systems' CJK fonts include both a half- and proportional-width set of Roman characters. Some font instances use half-width Roman, and some use proportional-width Roman. Which *H*, *x*, *d*, and *p* you use to calculate the *etmCapHeight*, *etmXHeight*, *etmLowerCaseAscent*, and *etmLowerCaseDescent* fields depends on which set of Roman characters the font instance uses in the one-byte range.

Some fonts, such as Adobe Systems' kana (subset) fonts or Adobe-Japan2-0 CID-keyed fonts, do not include half- or proportional-width Roman characters. (Adobe-Japan2-0 CID-keyed fonts are not usable on Windows at this time because Shift-JIS encoding does not support the JIS X 0212–1990 character set.) In such cases, reasonable values for *etmCapHeight*, *etmXHeight*, *etmLowerCaseAscent*, and *etmLowerCaseDescent* should be used. Because the two-byte characters in Adobe Systems' CJK fonts are optically centered between y = –120 and y = 880, these values are used as defaults. That is, 880 for *etmCapHeight*, *etmXHeight*, and *etmLowerCaseAscent*; and 120 (*not* –120; absolute value required) for *etmLowerCaseDescent*.

6 PFM PostScript Information

The three fields in this section are null-terminated strings that provide PostScript- or driver-related information, as shown in Table 4.

Table 4 *PostScript Information*

String Name	Data Size	Value
Device Type	Null-terminated string of bytes	The static string "PostScript" followed by a null byte

Table 4 *PostScript Information (Continued)*

String Name	Data Size	Value
Windows Name	Null-terminated string of bytes	Windows' menu name as it should appear in applications' font menus followed by a null byte
PostScript Name	Null-terminated string of bytes	Fully-qualified PostScript font name followed by a null byte.

Note that if the font is for vertical use (that is, its /WMode value is 1, or otherwise intended for vertical use), an "at" symbol ("@"; hexadecimal 0x40) *must* be prepended to the Windows Name string. This increases the length of the Windows Name string by one byte.

The *Windows Name* string is typically in localized non-ASCII script. For Japanese, it is a Shift-JIS string. For Korean, it is an EUC-KR string (but the font instance itself can specify UHC encoding, which is a superset of EUC-KR encoding). For Simplified Chinese, it is an EUC-CN string (but the font instance itself can specify GBK encoding, which is a superset of EUC-CN encoding). For Traditional Chinese, it is a Big Five string.

Because UHC and GBK encodings were supported starting with Windows 95, any characters specific to those encodings would not be recognized in pre-Windows 95 systems. This is why EUC-KR and EUC-CN encodings are recommended for Korean and Simplified Chinese, respectively.

The *PostScript Name* string is a fully-qualified PostScript font name (that is, a valid argument to the **findfont** or **selectfont** operators). For CID-keyed fonts, this means a CIDFont name plus a CMap name concatenated using one or two hyphens. Two hyphens are preferred, but one hyphen may be necessary for compatibility with fonts that had a previous life as an OCF (Original Composite Format) font. Some examples include the following:

STSong-Light--GBK-EUC-V (CID-only font)

Ryumin-Light-RKSJ-H	(a font that has existed as both an OCF and a CID-keyed font)

7 PFM Extent Table

This PFM table contains the number of fields specified by the range *dfFirstChar* to *dfLastChar* (32 to 255; 224 fields total). Each field is represented by two bytes, and indicates the width of each character in design units. For encoded values that represent the first byte of a two-byte character, the value should be set to 500. The total size of this table is 448 bytes.

8 The Order and Location of PFM Structures

The PFM Header and Extension must be located at the beginning of the PFM file, and must appear in that order. The exact order and location of the Extended Text Metrics section, Device Type string, Windows Name string, PostScript Name string, and Extent Table does not matter because offsets to these structures are explicitly set in earlier PFM fields. But, the convention is to order them in a consistent manner. Adobe Systems' convention is to include them in the following order:

Extended Text Metrics section
Device Type string
Windows Name string
PostScript Name string
Extent Table

Following the above order has the benefit of fixed values for the following PFM fields:

dfDevice	199
dfFace	210
dfExtMetricsOffset	147

9 Creating PFM Files

Appendix A provides a Perl program that builds a well-formed PFM file. Perl is a freely-available interpreted programming language available for most platforms. For information on obtaining Perl for a specific platform,

please refer to the following URL:

http://www.perl.com/perl/

The Perl program can be extracted as ASCII text then used with a Perl interpreter, or the algorithm can be studied and implemented in another software tool.

The Perl program requires a simple data file as standard input (STDIN), then writes as standard output (STDOUT) a well-formed PFM file. The format of the data file is illustrated by two examples shown in Appendix B. The following is an example command line:

% perl mkpfm.pl < *data-file* > *output.pfm*

where "data-file" is the input data file, and "output.pfm" is the PFM file.

10 Naming PFM files

While there is no standard naming convention for PFM files, the name must conform to the "8.3" Windows naming convention (an eight-character file name, a period, followed by a three-character extension). The extension must be "pfm".

11 Installing and Registering PFM Files

The installation of PFM files, including how to register them with the Windows operating system and Adobe Type Manager software, is described in Adobe Technical Note #5175, "Installing CID-Keyed Fonts for ATM Software." The URL for this and other Adobe Technical Notes is:

http://www.adobe.com/supportservice/
devrelations/technotes.html

Appendix A
Perl Program for Creating PFM Files

The following is a Perl program for constructing PFM files for PostScript-language CJK fonts.

```
#!/usr/local/bin/perl -w

# Version 1.0
# December 18, 1996
# Written by Ken Lunde, Adobe Systems Incorporated (lunde@adobe.com)

require 5.002;

$vertical = $errorcount = 0;
@six_fields = (100, 50, 50, 100, 25, 25);

@names = qw[dfCopyright dfAscent dfInternalLeading dfWeight dfCharSet
  dfPitchAndFamily etmCapHeight etmXHeight etmLowerCaseAscent
  etmLowerCaseDescent WindowsName PSName Widths];

# Read in the contents of the data file, ignoring blank lines and non-
# key/value pairs

while($line = <STDIN>) {
  $line =~ s/^\s+//;
  $line =~ s/\s+$//;
  next unless ($key,$value) = split(/\s*=\s*/,$line,2) and defined $value;
  $pfm{lc $key} = $value;
}

# First, check to make sure that each required key exists

foreach $key (@names) {
  if (!exists $pfm{lc $key}) {
    print STDERR "$key is undefined!\n";
    $errorcount++;
  }
}
die "Exiting...\n" if $errorcount;

# If the WindowsName field is in an eight-bit printable format, convert
# it to true eight-bit. Two lowercase or uppercase hexadecimal digits,
# which represent a single character, can be prefixed with either an
# equals (=) or a percent (%) sign.
```

```
$pfm{windowsname} =~ s/[=%]([\dA-Fa-f]{2})/pack("C",hex($1))/eg;

# Calculate offsets and file size

$driverinfooffset = 210 + length($pfm{windowsname}) + 1;
$extenttbloffset = $driverinfooffset + length($pfm{psname}) + 1;
$size = $extenttbloffset + 448;

# Determine whether the font is vertical

if ($pfm{windowsname} =~ /^\@/) {
  $vertical = 1;
  @six_fields = (0, 0, 0, 0, 0, 0);
}

&MakeExtentTbl;

# Below are the binary packing structures for each section of the PFM.
# "v" is a little-endian two-byte value, "V" is a little-endian four-
# byte value, "a" is null-padded ASCII text, "C" is an unsigned char
# (one-byte), and "x" is a null byte.

$pat1 = "vVa60vvvvvvvCCCvCvvCvvCCCCvVVVV"; # PFM Header
$pat2 = "vVVVVVVV";                        # PFM Extension
$pat3 = "vvvvvvvvvvvvvvvvvvvvvvvvvv";      # PFM Extended Text Metrics
$pat4 = "a*xa*xa*x";                       # PFM PostScript Information

# Each section is packed using the appropriate packing structure and
# values.

$string = pack($pat1,        # Packing structure for PFM Header
  256,                       # dfVersion
  $size,                     # dfSize
  $pfm{dfcopyright},         # dfCopyright
  129, 10, 300, 300,         # dfType, dfPoint, dfVertRes, dfHorizRes
  $pfm{dfascent},            # dfAscent
  $pfm{dfinternalleading},   # dfInternalLeading
  196,                       # dfExternalLeading
  0, 0, 0,                   # dfItalic, dfUnderline, dfStrikeOut
  $pfm{dfweight},            # dfWeight
  $pfm{dfcharset},           # dfCharSet
  0, 1000,                   # dfPixWidth, dfPixHeight
  $pfm{dfpitchandfamily},    # dfPitchAndFamily
  500, 1000, 32, 255,        # dfAvgWidth, dfMaxWidth, dfFirstChar, dfLastChar
  0, 0, 0,                   # dfDefaultChar, dfBreakChar, dfWidthBytes
  199, 210, 0, 0);           # dfDevice, dfFace, dfBitsPointer, dfBitsOffset

$string .= pack($pat2,       # Packing structure for PFM Extension
  30, 147,                   # dfSizeFields, dfExtMetricsOffset
  $extenttbloffset,          # dfExtentTable
  0, 0, 0,                  # dfOriginTable, dfPairKernTable, dfTrackKernTable
  $driverinfooffset,         # dfDriverInfo
  0);                        # dfReserved

$string .= pack($pat3,       # Packing structure for PFM Extended Text Metrics
  52, 240, 0,                # etmSize, etmPointSize, etmOrientation
  1000, 3, 1000,             # etmMasterHeight, etmMinScale, etmMaxScale
```

```
  1000,                      # etmMasterUnits
  $pfm{etmcapheight},        # etmCapHeight
  $pfm{etmxheight},          # etmXHeight
  $pfm{etmlowercaseascent},  # etmLowerCaseAscent
  $pfm{etmlowercasedescent}, # etmLowerCaseDescent
  0, -500, 250,              # etmSlant, etmSuperScript, etmSubScript
  500, 500,                  # etmSuperScriptSize, etmSubScriptSize
  @six_fields,               # etmUnderlineOffset, etmUnderlineWidth,
                             # etmDoubleUpperUnderlineOffset,
                             # etmDoubleLowerUnderlineOffset,
                             # etmDoubleUpperUnderlineWidth,
                             # etmDoubleLowerUnderlineWidth
  405, 50,                   # etmStrikeOutOffset, etmStrikeOutWidth
  0, 0);                     # etmKernPairs, etmKernTracks

$string .= pack($pat4, "PostScript", $pfm{windowsname}, $pfm{psname});

foreach $key (sort {$a <=> $b} keys %extenttbl) {
  $string .= pack("v", $extenttbl{$key});
}

# Now, write the PFM to STDOUT

print STDOUT $string;

# The following function builds a database for creating the Extent Table,
# which provides explicit width information for the characters in the
# range 32 through 255, in device units.

sub MakeExtentTbl {
  foreach $element (32 .. 255) {
    $extenttbl{$element} = 500;
  }
  if ($pfm{widths} =~ /^\d+$/) {
    foreach $element (32 .. 126) {
      $extenttbl{$element} = $pfm{widths};
    }
  } elsif ($pfm{widths} =~ /,/) {
    @widths = split(/\s*,\s*/,$pfm{widths});
    $count = 0;
    foreach $element (32 .. ($#widths + 32)) {
      $extenttbl{$element} = $widths[$count];
      $count++;
    }
  }
}
```

Appendix B
Sample Data Files for Creating PFM Files

The following data file is used for constructing PFM files for PostScript-language CJK fonts, using the Perl program provided in Appendix A. This example uses a single value for the *Widths* keyword, indicating that the one-byte Roman characters are all half-width (500 units wide).

```
dfCopyright=Copyright 1985-1997 Adobe Systems Inc.
dfAscent=880
dfInternalLeading=134
dfWeight=400
dfCharSet=134
dfPitchAndFamily=16
etmCapHeight=675
etmXHeight=447
etmLowerCaseAscent=704
etmLowerCaseDescent=195
WindowsName=华文宋体
PSName=STSong-Light--GBK-EUC-H
Widths=500
```

Two alternative methods for specifying the *WindowsName* string are as follows (eight-bit characters are represented by two hexadecimal digits prefixed with either an equals (“=”) or percent (“%”) sign):

```
WindowsName==BB=AA=CE=C4=CB=CE=CC=E5
WindowsName=%BB%AA%CE%C4%CB%CE%CC%E5
```

The following data file example illustrates the use of a comma-separated array to list width values for proportional one-byte Roman characters in the font. The array of width values must correspond to the encoded range 32 (0x20) through 126 (0x7E), namely 95 values.

```
dfCopyright=Copyright 1985-1997 Adobe Systems Inc.
dfAscent=880
```

```
dfInternalLeading=28
dfWeight=400
dfCharSet=129
dfPitchAndFamily=17
etmCapHeight=719
etmXHeight=478
etmLowerCaseAscent=727
etmLowerCaseDescent=141
WindowsName=HY신명조
PSName=HYSMyeongJo-Medium--KSCms-UHC-H
Widths=333,416,416,833,625,916,833,250,500,500,500,833,291,833,
  291,375,625,625,625,625,625,625,625,625,625,625,333,333,833,
  833,916,500,1000,791,708,708,750,708,666,750,791,375,500,791,
  666,916,791,750,666,750,708,666,791,791,750,1000,708,708,666,
  500,375,500,500,500,333,541,583,541,583,583,375,583,583,291,
  333,583,291,875,583,583,583,583,458,541,375,583,583,833,625,
  625,500,583,583,583,750
```

The alternative forms of the *WindowsName* string are as follows:

```
WindowsName=HY=BD=C5=B8=ED=C1=B6
WindowsName=HY%BD%C5%B8%ED%C1%B6
```