Microsoft® Windows
Version 2.1

Printer Fonts Kit

Microsoft Corporation

PREFACE

The *Microsoft Windows Printer Fonts Kit* contains documentation and software for third-party developers working on printer and screen fonts for use with Windows applications on the PC. Specific information is provided on how to make fonts compatible with the Microsoft Windows environment. The *Microsoft Windows Fonts Kit* is intended for use by hardware and software engineers.

The *Microsoft Windows Printer Fonts Kit* contains these two documents:

Microsoft Windows Fonts Guide

Microsoft Windows Technical Notes on the PCL Driver

The *Microsoft Windows Fonts Guide* contains information on generating a standard Microsoft Windows .PFM file plus details on .PFM files for PostScript and PCL printers. It also discusses screen fonts for Windows. The *Microsoft Windows Technical Notes on the PCL Driver* explains version 3.2 of the Windows PCL/HP LaserJet printer driver.

If you have questions regarding information in this document, please contact Microsoft Product Support Services.

Microsoft$_R$ Windows
Version 2.1

Fonts Guide

Microsoft Corporation

CONTENTS

CHAPTER 1

INTRODUCTION

CONTENTS

1 INTRODUCTION

This document discusses printer and screen font issues in the Microsoft® Windows environment. It is intended to be used by printer manufacturers and screen/printer font suppliers for the Windows environment.

1.1 Background

Laser printer output, which features a rich variety of fonts, has made today's personal computer users more aware of typography and quality output. Dot-matrix printer output, once considered acceptable for final copy, is now regarded as draft quality. Daisywheel printers, which use typewriter-like fonts, are also less acceptable. And new font offerings, such as those of Apple's LaserWriter Plus and the IBM Personal Pageprinter, promise to continue this trend toward typographic-quality output.

The success of the PC desktop publishing market, and to a lesser extent Microsoft Windows, will be greatly affected by the number and quality of fonts available for printers that serve that market. We believe that desktop publishing applications drive the sales of laser printers. The degree of success that a given printer will enjoy in the desktop publishing market will be greatly affected by the fonts available for that printer.

The same is also true for the word processing and marketing presentations markets, which are growing steadily more and more sophisticated in their use of fonts.

1.1.1 The Microsoft Windows Environment and the Future

The Microsoft Windows (and OS/2 Presentation Manager) operating environment is becoming the standard graphics environment for PC-compatible computers, particularly in the areas of WYSIWYG word processing and desktop publishing.

Print devices and device fonts that adhere to the Windows standard will be automatically supported by Windows application software. The needs of many applications can be addressed by a single investment in a Windows device driver and a set of matching WYSIWYG screen fonts. We believe that the return on such an investment will be quite dramatic.

1.1.2 Printer and Screen Font Fidelity

WYSIWYG screen representation of the document removes the limitations inherent in old-style composition packages making typographic-quality output possible by the non-typographer. WYSIWYG capabilities were once available only to users of expensive, dedicated systems. Now, however, they are becoming possible, and even expected, on most mass-market personal computers.

The shift from dedicated to mass-market personal computers has created opportunites for people other than printer manufacturers to provide the market with WYSIWYG screen fonts. The key is the development, acquisition, or licensing of efficient and reliable font-generation technology. With such technology, a retail font supplier could generate both printer and screen fonts in a wide variety of sizes quickly and inexpensively. Third parties who wish to provide screen fonts to match a particular printer's fonts should work closely with the printer manufacturer to ensure screen and printer font fidelity.

Font distributors who can provide both printer and matching screen fonts in the *same* retail package will be in an advantageous position in a marketplace where the majority of font suppliers provide only printer fonts.

1.2 Windows Font Requirements

Regardless of how the vendor distributes fonts, the Windows environment must ultimately receive these items for each font.

For downloadable fonts:

- An appropriate aspect-ratio, bitmap, screen font file in Windows .FON file format usable by Windows GDI
- The printer download file in printer-specific format
- A printer font metrics file in Windows .PFM file format usable by the printer's Windows printer driver

For cartridges:

- The font .FON file
- A .PCM file
- A cartridge

You must update the Windows initialization file, WIN.INI, to make the screen driver and printer driver aware of these fonts. Ideally, the printer driver will contain an installer that will read the font vendor's floppy disks and install these files. In the absence of such a utility, clear directions on how to copy files to the user's hard disk and how to add entries to WIN.INI are important. However, we encourage font suppliers to provide their own user-friendly installer as a way to add value to their offering. (The Windows Control Panel can install screen fonts; only downloadable and .PFM files need to be installed by the driver.)

1.2.1 Bitmap Screen Fonts

Printer fonts should be accompanied by bitmap screen fonts in the apppropriate aspect ratio. Minimally, you must supply line sizes that match the frequently used graphic arts point sizes (8, 9, 10, 12, 14, 18, and 24).

You measure screen fonts in line sizes (dots or lines per inch) rather than typographic point sizes. Specific line sizes of the fonts may be different from their nominal point sizes. For example, on a 96-line-per-inch high-resolution display, such as Micro Display System's Genius Display, the graphic arts point sizes listed above correspond to line sizes 10, 12, 13, 16, 18, 24, and 32.

The vendor should supply additional line sizes to support high-resolution devices and different magnifications of the page (e.g., Aldus PageMaker's views). For more information on screen fonts, see Chapter 2, "Screen Fonts for Windows," in this document and Chapter 7, "Installer Scenarios," in the *Microsoft Windows Technical Notes on the PCL Driver*.

1.2.2 Downloadable Fonts

Downloadable fonts are the most cost-effective way to distribute additional fonts for a printer device. This approach allows third-party vendors to create and/or distribute downloadable soft fonts for a device.

Downloadable fonts also allow users to access a larger variety of fonts than could be held in their printer's memory. This is possible because the driver selectively downloads the fonts only when needed.

Each downloadable font must have a font file in printer-specific format. This file is in the printer's font command language. If the Windows printer driver is to download the file, it must know where it resides on the user's hard disk. Chapter 3, "Printer Font Metrics (PFM) Files," describes the Windows font management mechanism.

1.2.3 Printer Font Metrics Files

Each downloadable printer font needs a printer font metrics (PFM) file in the Windows .PFM file format. The Windows printer driver uses the information in the .PFM file to prepare the width and kerning tables used by the application during composition.

A printer font metrics file shares a common header with a Windows screen font file (see the *Microsoft Windows Software Development Kit Programmer's Reference Manual*). The differences between the screen font file and the printer font metrics file occur after the common header. For a description of the contents of .PFM files, see Chapter 3, "Printer Font Metrics (PFM) Files," and, then, Chapters 4 and 5 for specific descriptions of .PFM files for PostScript and PCL printers.

CHAPTER 2

SCREEN FONTS FOR WINDOWS

CONTENTS

2 SCREEN FONTS FOR WINDOWS

This chapter presents the factors you should consider when creating screen fonts. We recommend that you provide at least two (three would be better) sets of screen fonts for the important display classes. We also recommend a range of screen line sizes, taking into account the trade-off between performance and WYSIWYG quality.

Windows applications should always compose text for the target output device and display an approximation of that output on the screen. To have a WYSIWYG display, the screen fonts must match the printer fonts. The more accurate the representation, the more WYSIWYG the display.

2.1 The Aspect-Ratio Classes of Display Devices

Actual dots-per-inch resolution is not a factor in determining the class of the device; only the aspect ratio is used.

Windows recognizes three aspect-ratio classes of display devices:

- 2:1
- 1.33:1 (4:3)
- 1:1

These classes roughly correspond to:

- The IBM Color Graphics Adapter (2:1)
- The IBM Enhanced Graphics Adapter (4:3)
- The new IBM Personal System/2 MCGA (model 30), VGA (models 50, 60, and 80), and 8514/a (models 50, 60, 70, and 80) (1:1)

If you pre-digitize the screen fonts, we suggest you distribute three sets of *each* font for the three aspect ratios. (Vendors wanting to distribute only two sets should exclude the 2:1 class, as it clearly represents old technology.)

2.2 Line Sizes are Not the Same as Point Sizes

Because of the variation in screen resolutions, screen font sizes are discussed in terms of screen line sizes, not typographic point sizes. Those familiar with the Macintosh environment have become used to line sizes and point sizes being the same. This is *not* true in the PC environment.

Windows defines a logical page size (that is, lines per inch) for each device it supports. When possible, this logical page size will closely match the physical size and aspect ratio of the device. Common screen lines-per-inch sizes are as follows:

- For a 2:1 aspect ratio, 96:48 (IBM Color Graphics Adapter)
- For a 1:1 aspect ratio, 96:96 (IBM Personal System/2), 108:108, and 144:144
- For a 4:3 aspect ratio, 96:72 (IBM Enhanced Graphics Adapter)

The equation for relating point size to equivalent screen line size is:

$$\text{screen line size} = \frac{\text{text point size} * \text{lines per inch}}{72}$$

Truncate the quotient on the right-hand side if the result is a fractional line size (it is always better to select a smaller size when an exact match is not possible).

In applications that support zooming, the equation must include a magnification factor. For example, Aldus PageMaker supports five views in addition to the "Actual size" view covered by the above equation. These are as follows:

- Fit in window (approximately 40% on an EGA)
- 50% size
- 75% size
- 200% size
- 400% size

The general form of the equation that PageMaker uses to select the screen font line size is:

$$\text{screen line size} = \frac{\text{text point size} * \text{lines per inch}}{72} * \frac{\text{view \%}}{100}$$

PageMaker chooses the line size of the screen font based on the text's specified point size, the screen's lines per inch, and the PageMaker view. Only on a 72-line-per-inch screen at the "Actual size" view does PageMaker choose the same size screen display font as the specified point size.

At other resolutions and/or views, the chosen size may be larger or smaller than the output text point size. For example:

- On a 108-lines-per-inch screen at the "Actual size" view, 12-point text is represented by choosing an 18-line screen font (12 * 108 / 72 = 18).

- On a 48-lines-per-inch screen at the "Actual size" view, the same 12-point text is represented with an eight-line screen font (12 * 48 / 72 = 8).

- On a 72-lines-per-inch screen at the 50% view, 12-point text is represented using a six-line screen font (12 * 72 * 50 / (72 * 100) = 6).

These factors indicate that the more line sizes available, the higher the probability the correct line size will be available. In other words, a rich range of screen line sizes improves the WYSIWYG quality of the display. However, there is a trade-off between performance and the number and size of screen fonts.

2.3 Choosing the Correct Range of Line Sizes

To accurately represent a printer font on the screen, you must make available a reasonable range of screen fonts. However, each screen font consumes valuable memory, so as a font vendor, you should carefully weigh performance against large screen-font variety. For each printer font, you should distribute at least two, preferably three, .FON files:

- One file containing screen fonts for 1:1 aspect-ratio screens
- One file containing screen fonts for 4:3 aspect-ratio screens
- One file containing screen fonts for 2:1 aspect-ratio screens (optional)

The most important factor to consider when creating screen-font files is memory consumption. Fewer and smaller screen fonts use less memory, but may degrade the WYSIWYG quality of the display. Other factors to consider include the following:

- **Legibility threshold.** There is a certain size at which the fonts are difficult, if not impossible, to read. For most screens, the value is 6 lines. For high-resolution screens, the value is around 9-10 lines. Therefore, it does not make sense to provide fonts below these sizes.

- **Doubling and tripling.** The screen driver may double or triple a small-size font to take the place of an unavailable large font. For example, it may double an 18-line font to make a 36-line font, or it may triple a 24-line font to make a 72-line font. Doubling is ugly, but acceptable and difficult to avoid. Tripling is unacceptable and can be avoided by providing a well-chosen range of screen line sizes.

- **Specific applications.** The application can control how the screen driver will select a font. Aldus PageMaker, for example, affects greeking, doubling and tripling, vector font substitution, and down-sizing to match widths.

2.3.1 Greeking

Certain applications, such as Aldus PageMaker, stop displaying screen fonts below a certain line size (PageMaker's default is 6) and, instead, display a gray shaded box. This is referred to as greeked text. Greeking speeds up the screen display.

2.3.2 Proof Versus Draft (Doubling and Tripling) Quality

In Windows, *proof* usually means good and *draft* means poor but quick. The screen driver will never double- or triple-size a font for proof quality. However, the screen driver will double- or triple-size smaller fonts to get the exact size it wants for draft quality. If the size is available, the screen driver will always choose it.

Doubling and tripling occurs only when the exact size is not available. For example, assume the only screen sizes are 10 and 19 lines, and the application wants a 20-line screen font. In proof quality, the screen driver would choose the 19-line font. In draft quality, the screen driver would double the 10-line font.

2.3.3 Vector Fonts

Vector outline fonts do not look as good as well-made raster screen fonts, but they look better than doubled or tripled screen fonts. Also, the amount of memory used by vector fonts is constant while the amount of memory used by raster fonts increases as the font size increases.

2.3.4 Down-sizing to Match Widths

When the screen fonts do not accurately represent the printer fonts (i.e., the height-to-width ratio on the printer font is different from that of the screen font), an application can ask for a smaller font to get the correct width. By doing this, the application can display the line more quickly and prevent the characters from overlapping.

This behavior can most often be observed when the font vendor does not supply matching screen fonts, or the screen fonts do not contain the same height-to-width ratio as their corresponding printer fonts.

2.4 Recommended Screen Font Sizes

Nothing produces better WYSIWYG than a wide range of screen font sizes. On the other hand, nothing degrades system performance faster than a large number of screen fonts. Therefore, the selection of screen fonts must balance desired WYSIWYG quality with system performance.

Suppose, for example, you have two machines with different screen font configurations. Machine A contains the screen font sizes 7, 10, and 16 lines. Machine B contains the sizes 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, and 18 lines. If the user creates a page using all the fonts from Machine B, the differences in sizes will be visible. However, the machine will load 12 screen fonts to display the page.

On Machine A, with doubling and tripling disabled, the machine displays the same range of sizes as follows:

- 7, 8, and 9 lines displayed with the 7-line font
- 10, 11, 12, 13, 14, and 15 lines displayed with the 10-line font
- 16, 17, and 18 lines displayed with the 16 line font

Much of the WYSIWYG would be lost because several sizes are displayed with the same font. However, the machine had to load only three fonts to display the page.

Thus, Machine B *slowly* displayed with high WYSIWYG quality, what Machine A *quickly* displayed with poor WYSIWYG quality.

As the font vendor, you need to consider the application the user will run under Windows. For example, Windows Excel generally uses body-size fonts (fonts in the size range of 8 to 12 points), so a wide range of font sizes is unnecessary. And Aldus PageMaker, although it uses a large range of sizes, switches to vector fonts above 24 lines (the default "Vector text above" setting). Therefore, it does not make sense to provide screen fonts above 24 lines unless you modify (or instruct the user to modify) PageMaker's "Vector text above" setting.

Ideally, the user should decide which is more important: WYSIWYG quality or performance. Many font products are available which allow for exactly that: the product contains a font generator which requires the user to specify which point sizes to build. For such programs, we recommend the user be given the option to select a pre-determined range of sizes:

- "Publishing" size range of 8, 9, 10, 12, 14, 18, and 24 lines
- "Artwork" size range of 7, 8, 9, 10, 11, 12, 14, 16, 18, 21, 24, 28, and 32 lines
- "General use" or "Performance" range of 8, 10, 12, 18, and 24 lines

If the font-creation program requires the user to indicate exactly which sizes should be built (that is, there are no pre-determined ranges), then we recommend that the documentation provide specific instructions on what numbers to enter, based upon the user's needs or intended use for the product.

For font packages that give the user no choice in the size range, we recommend that the vendor supply the "Publishing" size range of 8, 9, 10, 12, 14, 18, and 24 lines.

CHAPTER 3

PRINTER FONT METRICS (PFM) FILES

CONTENTS

3 PRINTER FONT METRICS (PFM) FILES

This chapter presents the format for listing printer font metrics (PFM) files and soft fonts in the WIN.INI file, and the format for .PFM and .PCM files.

Most drivers have the text metrics for a small number of device-resident fonts built in as resources. But the majority of the printer font metrics will be external to the driver in the form of Windows printer font metrics (PFM) files.

The driver retrieves information about which fonts are available by reading profile strings from the Windows initialization file, WIN.INI. Drivers maintain their default settings across invocations of the driver by saving the information in WIN.INI.

3.1 Listing .PFM Files in the WIN.INI

Most of this discussion pertains to .PFM files for soft fonts (fonts that must be downloaded to the printer from the computer). However, cartridge and printer-resident fonts can be handled in the same manner.

There are two forms of soft fonts:

1. **Permanent soft fonts**. These are copied from the computer to the printer when the printer is first turned on and remain there until the printer is turned off. The Windows printer driver has access to permanent soft fonts for all the documents it prints.

2. **Temporary soft fonts**. These are copied from the computer to the printer on an as-needed basis during the print job. They are deleted from the printer's memory at the end of a print job.

For the Windows driver to correctly locate .PFM files, the Windows printer driver, or a utility provided by the font vendor, must perform the following operations:

- Transfer .PFM, screen font, and downloadable font files (soft fonts) from the font manufacturer's floppy disks to the user's hard disk

- Add the entries in the WIN.INI file in the format described in the following sample listing of .PFM files:

```
[HPPCL,LPT1]
SoftFonts=4
SoftFont1=C:\PCLFONTS\HVPB0140.PFM
SoftFont2=C:\PCLFONTS\TRPR0120.PFM
SoftFont3=C:\PCLFONTS\TRPB0120.PFM,C:\PCLFONTS\TR120BPN.USP
SoftFont4=C:\PCLFONTS\TRPI0120.PFM,C:\PCLFONTS\TR120IPN.USP
Cartridge100=C:\PCLFONTS\WUMPUS.PCM
```

The Windows Soft Font Installer, which is part of the PCL / HP LaserJet driver, also requires these entries for permanently downloaded fonts:

```
C:\PCLFONTS\HVPB0140.PFM=C:\PCLFONTS\HV140BPN.USP
C:\PCLFONTS\TRPR0120.PFM=C:\PCLFONTS\TR120BPN.USP
```

The WIN.INI entries in the preceding examples have these meanings:

WIN.INI Entry	Definition
[HPPCL,LPT1]	The driver stores all its information in a special section in the WIN.INI file. The format for the heading is: [*driver name,port name*] This allows the driver to maintain different information for each printer port to which the driver is connected. For example, the printer on LPT1 may be a standard LaserJet incapable of handling soft fonts while the printer on LPT2 may be a LaserJet Series II.
softfonts=	The number of softfont*n*= entries.
softfont*n*=	The first two softfont*n*= entries list the soft fonts that the user indicated have been previously downloaded to the printer (that is, permanently downloaded fonts). For these fonts, only a Windows printer font metrics (.PFM) file is referenced. These entries could also represent .PFM files for cartridge fonts. The driver detects cartridge fonts by the presence of a printer escape in the .PFM file. This is described in the *Microsoft Windows Technical Notes on the PCL Driver*. The last two softfont*n*= fields list the fonts that the driver is to download dynamically (temporary fonts) if called for in the document.
cartridge*n*=	The definition of a cartridge. A .PCM file contains one PFM structure for each font in the cartridge.
pfm-file=	The Soft Font Installer in the PCL / HP LaserJet driver uses these entries to preserve the file names of the permanently downloaded fonts (first two softfont*n*= entries). The Soft Font Installer is described in the *Microsoft Windows Technical Notes on the PCL Driver*.

The softfonts= and softfont*n*= entries in the WIN.INI file may be entered by the user or, ideally, automatically entered by an installation utility. Some Windows drivers, such as the PCL / HP LaserJet driver, contain an installation utility.

If the driver does not contain an installation utility, we recommend the font vendor provide a utility that:

- Transfers .PFM, screen font, and printer font files from the font vendors' disks to the user's hard disk.
- Writes the necessary changes to the WIN.INI file.

If .PFM files are not supplied, the driver may elect to generate them from the download files. The PCL / HP LaserJet driver, for example, does do this. However, the PCL driver's PFM generator does not create pair-kern tables or very accurate extended text metrics structures; Only the font's designer can accurately create these structures. It also cannot decompress compressed font files or decrypt encrypted files. In this case, the font vendor must supply an installer utility that will convert the font files into regular HP soft font files.

The current Windows PostScript driver is not equipped with a soft font install utility. Even if it were, it would not attempt to generate .PFM files from downloadable font files due to the complexity of PostScript fonts. Instead, it would build .PFM files from Adobe font metrics (.AFM) files. As a font vendor, you must supply the .PFM files so you can guarantee an accurate description of the font metrics. Optionally, you may supply .AFM files as a future driver may also read them.

3.2 Soft Font Install Directory File

The soft font install directory file, SFINSTAL.DIR, is a standard file format read by soft font install utilities. The purpose of this file is to link each downloadable font file to its corresponding .PFM and screen font file. The utility uses this file to locate the .PFM, screen, and printer font files on the disks in a distribution package. We recommend that font vendors use this file format when designing a font package for installation by a Windows soft font installer.

As of this writing, the soft font install directory file is supported only by the Soft Font Installer in the Windows PCL / HP LaserJet driver. The format for the file is documented in the *Microsoft Windows Technical Notes on the PCL Driver*.

3.3 .PFM File Organization

The printer font metrics file consists of the following data structures (in this order):

1. .PFM header structure (PFMHEADER)
2. Array of character widths from *dfFirstChar* through *dfLastChar* plus one word.
3. .PFM extension structure (PFMEXTENSION)
4. Driver name pointed to by *dfDevice*
5. Windows font name pointed to by *dfFace*
6. Extended Text Metrics structure (EXTTEXTMETRICS) pointed to by *dfExtMetricsOffset*
7. Extent table pointed to by *dfExtentTable*
8. Driver-specific structure pointed to by *dfDriverInfo*
9. Pair-kern table pointed to by *dfPairKernTable*
10. Track-kern table pointed to by *dfTrackKernTable*

Of these structures, the PFM header, PFM extension, and font name must be present in every Windows .PFM file. The remaining structures exist depending upon the printer driver and the font vendor's preferences. The specific requirements for the Windows PostScript and PCL drivers are described in Chapters 4 and 5. The rest of this section describes the data structures in the .PFM file.

3.3.1 PFM Header and Width Table

The header for the printer font metrics file begins with the same information stored in a Windows raster font file or a vector font file (see the *Microsoft Windows Software Development Kit Programmer's Reference Manual* for descriptions of those fields).

The PFM file header contains the following fields:

```
WORD  dfVersion
WORD  dfSize
char  dfCopyright[60]
WORD  dfType
WORD  dfPoints
WORD  dfVertRes
WORD  dfHorizRes
WORD  dfAscent
WORD  dfInternalLeading
WORD  dfExternalLeading
BYTE  dfItalic
BYTE  dfUnderline
BYTE  dfStrikeOut
WORD  dfWeight
BYTE  dfCharSet
WORD  dfPixWidth
WORD  dfPixHeight
BYTE  dfPitchAndFamily
WORD  dfAvgWidth
WORD  dfMaxWidth
BYTE  dfFirstChar
BYTE  dfLastChar
BYTE  dfDefaultChar
BYTE  dfBreakChar
WORD  dfWidthBytes
DWORD dfDevice
DWORD dfFace
DWORD dfBitsPointer
DWORD dfBitsOffset
```

The fields in PFMHEADER are identical to the fields for Windows screen font files. They are documented in the *Microsoft Windows Software Development Kit Programmer's Reference Manual.*

The width table is an array of *dfLastChar* - *dfFirst* + 2 words. (The last extra entry is 0.) If present, it is placed between the PFMHEADER and PFMEXTENSION structure. There is a width table for variable-width PCL fonts, but none for PostScript .PFM files. See Chapters 4 and 5 for descriptions of driver-specific implementations.

The width table is present only in .PFM files for variable-pitch PCL fonts. Fixed-pitch PCL fonts do not require width information beyond the width fields in the header; PostScript fonts use the extent table instead.

3.3.2 PFM Extension Data Structure

The PFMEXTENSION structure is:

```
WORD  dfSizeFields
DWORD dfExtMetricsOffset
DWORD dfExtentTable
DWORD dfOriginTable
DWORD dfPairKernTable
DWORD dfTrackKernTable
DWORD dfDriverInfo
DWORD dfReserved
```

The fields are defined as follows:

Field	Description
dfSizeFields	The size in bytes of this structure.
dfExtMetricsOffset	The byte offset in the file to the EXTTEXTMETRIC structure for this font.
dfExtentTable	The byte offset in the file to the extent table. The size of the table is *dfLastChar* - *dfFirstChar* + 1 word. The extent table contains the unscaled widths (in 1/1000's of an em) of the characters in the font. This table must be present in PostScript .PFM files (see Chapter 4, ".PFM Files for PostScript Printers").
dfOriginTable	The byte offset in the file to the table of character origins. The size of the table is *dfLastChar* - *dfFirstChar* + 1 word. For screen font files, this table gives the signed amount of left adjustment to add to the current position before drawing the characters. Negative values indicate left movement; positive values indicate right movement. This should be NULL for printer font metrics files.
dfPairKernTable	The byte offset in the file to the optional pair-kerning table. This should be NULL if no kerning pairs are defined for this font. The size of the table is given by the *etmKernPairs* member of the EXTTEXTMETRIC structure.
dfTrackKernTable	The byte offset in the file to the optional track-kerning table. This should be NULL if no kerning tracks are defined for this font. The size of the table is given by the *etmKernTracks* member of the EXTTEXTMETRIC structure.
dfDriverInfo	The byte offset in the file to additional driver-specific information. Each driver writer is responsible for documenting the format of the data pointed to by this member. These formats will be made public so that third parties wishing to create compatible printer font metrics files may do so. For the Windows PostScript and PCL drivers, these structures are described in Chapters 4 and 5.
dfReserved	Reserved; must be NULL.

3.3.3 Extended Text Metrics Structure

The EXTTEXTMETRIC structure contains these fields:

```
typedef struct{
        short etmSize;
        short etmPointSize;
        short etmOrientation;
        short etmMasterHeight;
        short etmMinScale;
        short etmMaxScale;
        short etmMasterUnits;
        short etmCapHeight;
        short etmXHeight;
        short etmLowerCaseAscent;
        short etmUpperCaseDescent;
        short etmSlant;
        short etmSuperScript;
        short etmSubScript;
        short etmSuperScriptSize;
        short etmSubScriptSize;
        short etmUnderlineOffset;
        short etmUnderlineWidth;
        short etmDoubleUpperUnderlineOffset;
        short etmDoubleLowerUnderlineOffset;
        short etmDoubleUpperUnderlineWidth;
        short etmDoubleLowerUnderlineWidth;
        short etmStrikeOutOffset;
        short etmStrikeOutWidth;
        WORD  etmKernPairs;
        WORD  etmKernTracks;
        }EXTTEXTMETRIC;
```

The EXTTEXTMETRIC fields are described below. All the measurements are given in the specified units regardless of the current mapping mode of the display context.

Field	Definition
etmSize	The size of this structure in bytes. Future changes to this structure will be additions to the existing structure; the position and meaning of existing members will not change.
etmPointSize	The nominal point size of this font in twips (twentieths of a point). This is the intended graphic arts size of the font; the actual size may differ slightly depending on the resolution of the device.

Field	Definition
etmOrientation	The font's orientation: • 0 if either orientation • 1 if portrait • 2 if landscape This value refers to the ability of this font to be imaged on a page of the given orientation. Landscape pages are those whose width (x-dimension) is greater than their height (y-dimension).
etmMasterHeight	Specifies the font size in device units for which the values in this font's extent table are exact. This is described in detail in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits".
etmMinScale	Specified in device units, the minimum valid size for this font. That is, $\text{smallest point size} = \frac{\textit{etmMinScale} * 72}{\textit{dfVertRes}}$ Where: 72 represents the number of points per inch. *dfVertRes* contains the number of dots per inch.
etmMaxScale	Specified in device units, the maximum valid size for this font. That is, $\text{largest point size} = \frac{\textit{etmMaxScale} * 72}{\textit{dfVertRes}}$ Where: 72 represents the number of points per inch. *dfVertRes* contains the number of dots per inch.
etmMasterUnits	The integer number of units per em where an em equals *etmMasterHeight*. In other words, *etmMasterHeight* expressed in font units rather than device units. This is described in detail in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits."
etmCapHeight	The height in font units of uppercase characters in the font. Typically, the height of the capital H.
etmXHeight	The height in font units of lowercase characters in the font. Typically, the height of the lowercase x.
etmLowerCaseAscent	The distance in font units that the ascender of lowercase letters extends above the baseline, typically specified for a lowercase d.
etmLowerCaseDescent	The distance in font units that the descender of lowercase letters extends below the baseline, typically specified for a lowercase p.

Field	Definition
etmSlant	The angle in tenths of degrees clockwise (to the right) from the upright version of the font (assuming this font is slanted or italicized).
etmSuperScript	The recommended amount in font units (see the discussion of font units in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits") to offset superscript characters from the baseline. This is typically a negative offset.
etmSubScript	The recommended amount in font units (see the discussion of font units in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits") to offset subscript characters from the baseline. This is typically a positive offset.
etmSuperScriptSize	The recommended size in font units of superscript characters for this font.
etmSubScriptSize	The recommended size in font units of subscript characters for this font.
etmUnderlineOffset	The offset in font units downward from the baseline where the top of a single underline bar should appear.
etmUnderlineWidth	The thickness in font units of the underline bar.
etmDoubleUpperUnderlineOffset	The offset in font units downward from the baseline where the top of the upper double underline bar should appear.
etmDoubleLowerUnderlineOffset	The offset in font units downward from the baseline where the top of the lower double underline bar should appear.
etmDoubleUpperUnderlineWidth	The thickness in font units of the upper underline bar.
etmDoubleLowerUnderlineWidth	The thickness in font units of the lower underline bar.
etmStrikeOutOffset	The offset in font units upward from the baseline where the top of a strike-out bar should appear.
etmStrikeOutWidth	The thickness in font units (see the discussion of font units in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits") of the strike-out bar.
etmKernPairs	The number of character kerning pairs defined for this font. You can use this value to calculate the size of the pair-kern table returned by the GETPAIRKERNTABLE escape. It will not be greater than 512 kern pairs.
etmKernTracks	The number of kerning tracks defined for this font. You can use this value to calculate the size of the track-kern table returned by the GETTRACKKERNTABLE escape. It will not be greater than 16 kern tracks.

3.3.4 Font Scaling: etmMasterHeight and etmMasterUnits

This section describes how a driver uses the values *etmMasterHeight* and *etmMasterUnits* to scale the character width values in the extent table. It also describes how the mechanism *should* work; Chapters 4 and 5 describe how each driver implements or intends to implement the mechanism.

A driver allows two methods for measuring character widths:

- Device units or "pixels"
- Font units

Device units are the number of dots in the physical device (dots per inch); *etmMasterHeight* is expressed in device units. It may be converted to points using the formula:

$$\text{height in points} = \frac{\textit{etmMasterHeight} \text{ device units} * 72 \text{ points per inch}}{\textit{dfVertRes} \text{ device units per inch}}$$

Where:
72 represents the number of points per inch.
dfVertRes contains the number of dots per inch in device units.

Font units may differ from device units as a convenience to the font vendor. If the font is targeted for devices that differ in resolution, there typically is no benefit from tying the resolution of the font to the resolution of a particular printer. The vendor uses the units that are most convenient for designing the font. For example, 1000 units per cell or em is the Adobe standard for PostScript fonts; *etmMasterUnits* is expressed in font units per em where an em equals *etmMasterHeight*.

A number may be converted from device units to font units using the formula:

$$\text{value in font units} = \frac{\text{value in device units} * \textit{etmMasterUnits} \text{ font units}}{\textit{etmMasterHeight} \text{ device units}}$$

The formula for the inverse operation is:

$$\text{value in device units} = \frac{\text{value in font units} * \textit{etmMasterHeight} \text{ device units}}{\textit{etmMasterUnits} \text{ font units}}$$

Determining Character Widths

The values in the extent table are the widths of the characters when the font height equals *etmMasterHeight*. When the font height does not equal *etmMasterHeight*, the driver must scale the values from the extent table (we always assume linear scaling fonts).

The values in the extent table are in font units. The width of a character for a particular font height may be computed by using the ratio of character extent over *etmMasterUnits*. That is:

$$\frac{\text{character width}}{\text{font height}} = \frac{\text{value from extent table in font units}}{\textit{etmMasterUnits} \text{ font units}}$$

By multiplying font height through the equation, we have the formula for computing character width:

$$\text{character width} = \frac{\text{font height} * \text{value from extent table in font units}}{\textit{etmMasterUnits} \text{ font units}}$$

Notice that character width takes on the units of font height. If the font height is in device units, the resultant width will be in device units. If the font height is in font units, the resultant width will be in font units.

In its normal mode of operation, the driver simply plugs the font height into the above equation to compute the character width. Font height is normally in device units, so the resultant width is normally in device units.

For example, assume the device resolution is 300 dpi and the font units are expressed as 1000 units per em. Assuming that *etmMasterHeight* represents a 72-point font, the values would look like this:

dfVertRes = 300
etmMasterHeight = 300
etmMasterUnits = 1000

Furthermore, assume the value in the width table for a capital H is 500 font units, or half the em height. Assume the height of the font for which we want the character width is 12 points, or 50 device units. With relative widths disabled, the character width would be computed using the first width formula:

$$\frac{\text{50 device units} * \text{500 font units}}{\text{1000 font units}} = \text{25 device units}$$

Using ENABLERELATIVEWIDTHS

When the application calls the ENABLERELATIVEWIDTHS escape, the driver expects the application to request a font height in device units, but will return the character width in font units. The application may want to obtain widths in font units to alleviate any error that may result from rounding widths to device units. It is the responsibility of the driver to correct for this error when relative widths are disabled, but the application may choose to enable relative widths and perform its own error correction.

The driver will convert the font height (in device units) to font units prior to computing the character width. The resultant formula is:

$$\text{character width in font units} = \frac{\text{font height in device units} * \text{character extent in font units}}{\textit{etmMasterHeight} \text{ device units}}$$

This formula was computed by replacing the "font height" from the first formula for character width with the formula for converting a value from device units to font units and, then, simplifying the equation.

Using the same assumptions given in the previous subsection, we can now work through some more examples. We know that the width of the character is 25 device units (i.e., dots or pixels) or half the height of 50 device units. With relative widths enabled, the character width would be computed using the second width formula:

$$\frac{\text{50 device units} * \text{500 font units}}{\text{300 device units}} = \text{83.333 font units}$$

The width of the character is 83.333 font units. The driver will round non-integer results.

To demonstrate that the width in font units is equivalent to the width in device units, it can be converted to device units using the formula provided in the previous subsection:

$$\frac{83.333 \text{ font units} * 300 \text{ device units}}{1000 \text{ font units}} = 25 \text{ device units}$$

As stated earlier, this section describes how *etmMasterUnits* and *etmMasterHeight* should be used. Chapters 4 and 5 describe restrictions imposed by the drivers.

3.3.5 Kern Pair Data Structure

The KERNPAIR structure contains the following fields:

```
typedef struct{
        union {
                BYTE each[2];  /* UNION: 'each' and 'both' share the same memory */
                WORD both;
                } kpPair;
        short   kpKernAmount;
        } KERNPAIR;
```

The KERNPAIR fields are described in the following table. All the measurements are given in the font units of the device regardless of the current mapping mode of the display context.

Field	Definition
kpPair.each[0]	The character code for the first character in the kerning pair.
kpPair.each[1]	The character code for the second character in the kerning pair.
kpPair.both	A WORD type in which the first character of the kerning pair is in the low-order byte and the second character is in the high-order byte.
kpKernAmount	The signed amount that this pair will be kerned if they appear side by side in the same font and size. This value is typically negative since pair-kerning usually results in two characters being set more tightly than normal.

The number of kern pairs is stored in the extended text metrics structure, *etmKernPairs*. The array of KERNPAIR structures returned by the GETPAIRKERNTABLE escape must be sorted in increasing order by the *kpPair.both* member (or in order of *kpPair.each[1]*, *lpPair.each[0]*. However, an integer sort on *kpPair.both* is faster). This allows the application to do a binary search on the kerning table.

3.3.6 Kern Track Data Structure

The KERNTRACK structure contains these fields:

```
typedef struct
        short ktDegree;
        short ktMinSize;
        short ktMinAmount;
        short ktMaxSize;
        short ktMaxAmount;
        }KERNTRACK;
```

The KERNTRACK fields are described below. All measurements are given in font units of the currently selected font regardless of the current mapping mode of the display context.

Field	Definition
ktDegree	A short integer value where increasingly negative values represent tighter track-kerning and increasingly positive values represent looser track kerning.
ktMinSize	A short integer specifying in device units the minimum font size for which linear track-kerning applies.
ktMinAmount	A short integer specifying in font units the amount of track-kerning to apply to font sizes *ktMinSize* and below.
ktMaxSize	A short integer specifying in device units the maximum font size for which linear track-kerning applies.
ktMaxAmount	A short integer specifying in font units the amount of track-kerning to apply to font sizes *ktMaxSize* and above.

Between the *ktMinSize* and *ktMaxSize* font sizes, track-kerning is a linear function from *ktMinAmount* to *ktMaxAmount*.

3.4 .PCM Files

A .PCM (Printer Cartridge Metrics) file consists of a global information header followed by a series of .PFM file structures. The structure is as follows:

```
struct _pcmheader {
        WORD pcmMagic;
        WORD pcmVersion;
        DWORD pcmSize;
        DWORD pcmTitle;
        DWORD pcmPFMList;
        }
```

The pcmMagic field must contain the magic number 3244 (0x0CAC). The Soft Font Installer uses it to recognize .PCM files when no SFINSTAL.DIR file is supplied with the .PCM file.

The pcmVersion field contains the version number of the .PCM file. The upper byte contains the version number and the lower byte the revision number, both in BCD. The current version number is 3.10, represented as 0x0310.

The pcmSize field contains the size in bytes of the entire .PCM file.

Following this structure is the title string for the cartridge. The file offset of this string appears in the pcmTitle field of the header. This string is terminated by zero. This title is normally used in the cartridge list box in the Printer Setup (device-mode) dialog box. A non-zero offset and a non-empty name string at that offset is required.

The list of font PFMs is a collection of .PFM files (one for each cartridge font) that are appended together. Each PFM contains (in a doubleword at offset 2) the size of the .PFM file, which you can consider the offset of the next PFM relative to the current one. All file positions in the PFM itself are offsets from the file position of the beginning of the individual PFM. The absolute file offset of the first PFM in the file is placed in the pcmPFMList field. One .PFM file is required for each individual font in the cartridge.

3.4.1 WIN.INI Additions

You should specify non-driver resident cartridges by adding a switch similar to the following one to the WIN.INI file in the driver section:

```
[HPPCL, port]
cartridgen=PCM-file
```

where the given .PCM file defines the cartridge with index *n*. You may use this index with the cartindex*n* switch to select the cartridge. The index of a cartridge must be at least 50 in version 3.2.

The cartridges defined by the cartridge*n* switch are listed in the Cartridge list box in the Printer Driver dialog box. You may select and unselect them as if they were standard cartridges.

3.4.2 Cartridge Installation

You install cartridges the same way as soft fonts by clicking the Fonts button to invoke the Soft Font Installer. All the user-defined cartridges that appear in the WIN.INI file are listed, along with the soft fonts, in the Fonts list box. Cartridges are labeled with a "(c)" symbol.

You can also add new cartridges the same way as fonts by clicking the Add Fonts... button and selecting a source from which to read. There are two ways to find cartridges. You can either specify them in the SFINSTAL.DIR file or scan the disk. The syntax for specifying cartridges in SFINSTAL.DIR is given in the next subsection. If no SFINSTAL.DIR file is found on the source disk, the scanning process will search for files in the PCM format (i.e., containing valid pcmMagic and pcmVersion fields and a valid title) and place those it finds in the install list box along with any downloadable fonts it finds.

You can manipulate cartridges basically the same way you would manipulate soft fonts. You can add, delete, copy, and/or move them between ports. However, the Edit button and the Permanent and Temporary radio buttons are disabled if any cartridges are selected since these operations do not apply to cartridges.

3.4.3 SFINSTAL.DIR Syntax

We have added a new structure for installing cartridges to the SFINSTAL.DIR syntax. The syntax is as follows:

```
CARTRIDGE {
        aspectratio = description, screenfontfile
                .
                .
                .
        "cartridge title" = cartridgefile
                .
                .
                .
        }
```

This syntax allows you to install matching screen fonts along with a cartridge. More than one cartridge may appear in a CARTRIDGE block. The *cartridge title* is the name of the cartridge placed in the list box at installation time. It should, but does not need to, match the title in the .PCM file.

CHAPTER 4

THE PFM EDITOR

CONTENTS

4 THE PFM EDITOR

One of the responsibilities of a Windows printer driver is to provide a list of the fonts available on the printer. Because of the large number of downloadable and cartridge fonts available for HP LaserJet printers, the driver can only have a few of these fonts built into it.

The driver supports the addition of known fonts by incorporating a utility called the Soft Font Installer. (See the *Microsoft Windows Technical Notes on the PCL Driver* for a description of this utility.) The Soft Font Installer, in the PCL driver, requires one .PFM file per font per size, with an additional .PFM file if there is a font variation such as an italic font, or a font with a different weight. Thus, a font cartridge containing Helvetica and Times Roman fonts at 8, 10, 12, and 14 points will require eight .PFM files. If the cartridge also contains boldface and italic versions of the 10 and 12 point fonts, an additional eight .PFM files are required. These individual cartridge font .PFM files are then combined into a single .PCM file.

The information Windows requires about each font includes identifying information such as the face name, family, and point size; size information such as heights, extents, leading, and widths of individual characters; information on how to synthesize effects such as underlining and strikeout; and hardware information such as the amount of printer memory required (for downloadable fonts) and the printer escape string used to select the font.

The font supplier generally provides .PFM files on the soft font disks or a .PCM file on a disk with the cartridge. The font supplier may optionally supply matching WYSIWYG screen fonts, which greatly enhance the usability of the fonts with Windows applications.

The PFM Editor is a Windows application that the supplier can use to create and edit .PFM files and create .PCM files for the PCL driver. The PFM Editor simplifies and accelerates the task of creating .PFM files by not requiring a great deal of knowledge about either the .PFM file format or Windows device drivers.

The PFM Editor will read the filename of a .PFM file from the command line if it is passed one. Thus, defining the .PFM extension in WIN.INI will allow the user to double-click .PFM files to edit them.

4.1 The Main Window

The main window of the PFM Editor contains several fields that give general identifying information about the font. All of these fields appear in the PFM header structure, which is identical to a font file header. Details of the field definitions appear in the SDK documentation. **(Where, Betsy? Currently, Sec. 7.2, "File Formats," of Programmers Reference?)** You should always specify at least the Pitch field before selecting any of the metrics or tables dialogs described in the following subsections.

The fields appearing in the main window are as follows:

Field	Description
Face Name	The face name of the font such as Helv or TmsRmn.
Char Set	The character set that the font represents. The most common values are 0 for Windows ANSI and 255 for the OEM character set. In the *Microsoft Windows Technical Notes on the PCL Driver*, additional character sets for the PCL driver are defined in Section 6.4.1, "PFM Header," as follows: 180 = Math-8; 181 = PI Font; 182 = Line Draw; 183 = PC Line; 184 = Tax Line, and 185 = US Legal.
Pitch	Two radio buttons: Fixed and Proportional.
Effects	Three options indicating the appearance of the font: Italic, Underline, Strikeout.
Font Family	Six selections: Don't Care, Roman, Swiss, Modern, Script, and Decorative. The driver uses the family to classify fonts for identification and selection purposes. The terms are documented in the Windows Software Development Kit. **(Where, Betsy? Currently, Sec. 7.2, "File Formats," of Programmers Reference.)**
Points	The point size of a font, or its height in units of 1/72 of an inch.
Weight	The weight of the font - a number from 0 to 1000. 400 represents a normal, medium font. Larger numbers represent heavier (bolder) fonts. A more exact correspondence appears in the *Microsoft Windows Technical Notes on the PCL Driver* under Section 6.4.1, "PFM Header."
Copyright	A 60-character string containing the font supplier's copyright notice.

4.2 The File Menu

The File menu contains all the standard Windows File menu items along with one extra item: Create PCM File... . The following is a list with descriptions of those items:

Field	Description
New	Causes the PFM Editor to re-initialize itself and display an empty, untitled .PFM file.
Open...	Displays a standard Open File Name dialog box. ".PFM" is the default file extension.
Save	Saves the .PFM file with the same name as previously used in a Load or Save operation. If the file is untitled, the Save As... dialog appears.
Save As...	Allows the user to specify a filename before saving.
Create PCM File...	Allows the user to create a cartridge metrics file for the HPPCL driver. For a more detailed description, see Section 4.6, "Creating .PCM Files."
Exit	Exits the PFM Editor.
About...	Displays an informational dialog box.

As with all applications, the user is given the opportunity to save or cancel if New, Open, or Exit is invoked when the current file has been saved but not changed.

4.3 The Metrics Menu

This menu allows you to activate dialog boxes to specify dimensional information about the font. The Metrics popup menu contains three selections: Basics..., Extended..., and Effects... .

Basic metrics includes information about character range heights and average widths, device resolution, leading, etc.

Extended metrics is detailed information about the geometry of the font, including such things as baseline, ascents/descents, orientation, scale, etc.

Effects metrics is information about how to synthesize effects such as underlining and strikeout. To get to this menu, though, you must first go through the Extended Metrics dialog box.

4.3.1 The Basic Metrics Dialog Box

You access this dialog by choosing Basics... from the Metrics menu. It contains information about font size. The dialog box fields are as follows:

Field	Description
Characters:	
First	The first character in the character set. This is the numeric character code of the first character for which the font contains a printable character.
Last	The last character in the character set. This is usually 127 for a 7-bit character set or 255 for an 8-bit character set.
Default	The code of the character printed if the driver is asked to print a character that falls outside the range First through Last.
Break	The character that delimits words in word wrapping. If an application or the DrawText() function is asked to word wrap a string in a box, it will break the word into multiple lines only at break character positions. This is generally the space character (32 in both ANSI and ASCII).
Leading:	
Internal	The leading that appears within the height specified by the height of the font. Diacritical marks appear in this space.
External	The recommended additional space to insert between lines. This space is not already accounted for in the height of the font.
Resolution:	
Horizontal and *Vertical*	The number of dots per inch at which the font was digitized. For an HP LaserJet, for example, both of these numbers should be 300.
Widths:	
Average	For proportional space fonts, the average width of all the characters in the font. This field is relabelled All for fixed pitch fonts, since all fixed pitch characters have the same width. It is represented in font units. The PFM Editor does not calculate the average width from the width table; it must be explicitly entered. It is generally the width of the lower case "x."
Maximum	The width in font units of the widest character in the font. Again, this is not calculated from the user width table. For fixed pitch fonts, this field is not used and is disabled.
Height	The height in font units of the character cell, including internal leading.
Ascent	The ascent in font units of the font. The distance from the top of the character cell to the baseline of the font.

With the exception of the width table for proportional fonts, this dialog fills in the remainder of the Microsoft-defined font file header. The field definitions are from the SDK. **(Where, Betsy?)** If the .PFM file is to be of any use, all of these fields must be accurately specified.

There are also two pushbuttons: OK, which causes the changes to be committed into the .PFM file, and Cancel, which rolls the .PFM file back to the state in which it was when the user invoked the menu selection.

4.3.2 The Extended Metrics Dialog Box

The extended font metrics are a more detailed description of the font's geometry. The major fields are as follows:

Field	Description
Lower Case:	
Ascent	The distance in font units from the baseline to the top of lower case ascenders, typically measured from the lower case "d."
Descent	The distance in font units from the baseline to the bottom of descenders, usually specified for lower case "p."
Point Size	The intended size of the font in twips (1/20th of a point, or 1/1440 of an inch).
Cap Height	The height in font units of the font's uppercase characters, usually taken for the character "H."
X Height	The height in font units of lower case characters, usually the height of the lower case "x."
Slant	The angle in tenths of a degree clockwise from the vertical.
Master Height	The size in device units at which the values in the extent table are exact.
Master Units	The Master Height field expressed in font units. If the two are different, then the device is a linear scaling device and the application needs to scale all values by the ratio of the requested size to this value. (See Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits," for a detailed description of these two fields.)
Scale:	
Min and *Max*	The minimum and maximum sizes to which a linear scaling device can scale a font.
Orientation:	
Portrait and *Landscape*	Specify the orientation of the font, which may be either or both.

The Master Height, Master Units, and Min and Max Scale fields are used on devices that can scale fonts linearly to any size. On devices that do not scale fonts, these numbers should all be equal to the value in the Height field from the Basic Metrics dialog box.

All these fields come from the Extended Text Metrics structure, which was defined in Section 3.3.3, "Extended Text Metrics Structure."

4.3.3 The Effects Metrics Dialog Box

This menu item is disabled (grey) until you specify the extended metrics. The information you can then specify in this dialog box tells the driver how to synthesize effects such as underlining and superscripting while allowing for the per-font, per-size customization of these effects. It also forms the remainder of the Extended Text Metrics structure in the .PFM file.

The effects of interest are superscripting, subscripting, underlining, double underlining, and strikethroughs. Two numbers are required: the vertical position of the effect and its size. Offsets are measured downward from the baseline of the font, which means that strikeouts and superscripts will generally have negative offsets. The size specifies the thickness of lines in font units for the line effects, or the height in font units for super- and sub-scripts. For double underlines, you may specify the upper and lower lines independently.

4.4 The Tables Menu

This popup menu contains three selections: Widths..., Kerning Pairs..., and Kerning Track... . Since these are only appropriate for proportional width fonts, they are grayed for fixed pitch fonts.

The width table contains the exact specification, per character, of the widths of characters in proportional fonts. Track and pair kerning is used to fine tune character spacing.

4.4.1 The Width Table Dialog Box

The Width Table dialog box has a scrollable list box containing all the characters and their widths, represented by decimal numbers in font units. Notice that, since this table is an absolute requirement for all proportional fonts, you must set the font pitch to proportional and the first and last characters in the Basics... dialog box before entering a width table.

Just above the list box is a label number with an edit control next to it. Whenever a character is selected from the list box, the label changes to the character's code, and the edit control is changed to its width. The width can then be changed.

To commit the width change to the width table in the listbox, you must click on the Width pushbutton. Since Width is the default button in this dialog box, you can also just press the ENTER key after working in the edit control box. If the character is not the last one in the font, clicking on Width or pressing the ENTER key also causes the selection to be advanced to the next character, which allows you to enter quickly the entire width table.

4.4.2 The Pair Kerning Table Dialog Box

You can fine tune character spacing for proportional fonts by using pair kerning. The Pair Kerning Table dialog box consists of a list box containing character pairs and the amount of space to insert or delete between them. A negative kern amount (given in font units) specifies tighter spacing, while a positive amount spreads characters out.

The three edit controls are for the From and To characters and the kerning amount. If you select a kern pair in the list box, the editor displays for modification the values in the structure.

There are three buttons for manipulating the Pair Kerning Table:

- **Add**, which adds to the table the kern pair in the edit controls.
- **Delete**, which removes from the list box the selected kern pair
- **Change**, which replaces the selected kern pair with the contents of the edit control.

Notice that the Pair Kerning Table is sorted; adding or replacing a pair does not give it a particular position in the table.

As in the other dialog boxes, the OK button causes the table to be committed to the .PFM file, and the Cancel button will end the dialog without saving the changes to the table.

4.4.3 The Track Kerning Table Dialog Box

You enter values into the Track Kerning Table in the same manner as for the Pair Kerning Table. There are five fields to consider.

For the kerning degree, negative values specify tighter kerning, and positive values specify looser kerning. The minimum size specifies the smallest size in font units for which track kerning applies, and the maximum size specifies the largest font.

The minimum amount specifies the amount of track kerning to apply to fonts of the minimum size and below. The maximum amount is the track kerning to apply to all fonts at least as large as the maximum size.

4.5 The Driver Menu

The PCL printer is currently the only supported printer in the Driver menu. The PCL Driver dialog box contains information specific to the PCL driver for HP LaserJet-type printers. The fields are as follows:

Field	Description
Symbol Set	Specifies the HP-defined symbol set for a font. (See the HP LaserJet Printer Technical Reference Manual for definitions of these symbol sets.) The following are the currently defined symbol sets (selectable with individual radio buttons): User Defined Roman 8 Kana 8 Math 8 U.S. ASCII Line Draw Math Symbols U.S. Legal Roman Ext. E.C.M.A. 94 ISO Den/Nor ISO U.K. ISO France ISO Germany ISO Italy ISO Sweden/Finland ISO Spain Generic 8 Generic 7
Mem Usage	An approximation of the amount of memory in the printer that the soft font requires. This is given in Section 6.4.3, "Device-Specific Structure," of the *Microsoft Windows Technical Notes on the PCL Driver* as the following formula: ((sum of all character widths + 7)>>3) * height + 63
Escape String	This is the escape string you send to the printer to select the font. It may contain any PCL commands. You may use the following special sequences to enter control characters: \e - escape (ASCII 27, 0x1b) \[- escape (ASCII 27, 0x1b) \n - line feed (ASCII 10, 0xa) \r - carriage return (ASCII 13, 0xd) \xnn - hexidecimal character code \nnn - octal character code Notice that when the PFM Editor displays an escape string, it converts all non-ASCII printing characters to the hexidecimal format \xnn.

As in the other dialog boxes, the OK button causes the items to be committed to the .PFM file, and the Cancel button ends the dialog without saving changes.

4.6 Creating .PCM Files

The PCL driver uses printer cartridge metric (PCM) files to define cartridges other than built-in Hewlett-Packard cartridges. .PCM files are essentially a collection of .PFM files (one for each font in the cartridge).

To make a .PCM file, first create the individual .PFM files for each of the cartridge fonts, using the PFM Editor features described in the preceding sections of this chapter. After you have created all the .PFM files and saved them to disk in one directory, choose the File menu's Create PCM File... option to have a dialog box appear and prompt you for a .PCM filename. Put this file in the same directory as its constituent .PFM files.

After you enter a filename and click on the OK button, a second dialog will appear with the .PCM filename in a static control at the top. The first edit control in the dialog box allows you to enter the cartridge title, which must be a non-empty title (i.e., you must fill in the edit control). This title is the string placed in the cartridge selection list box in the driver's Printer Setup (device-mode) dialog box.

The PFM Editor will place a list of .PFM files, that are in the same directory as the target .PCM file, in a list box on the left side of the dialog box. You may select one or more of these files and move them to the right-hand "selected" list box by clicking the Select button (or double-clicking a filename). You can remove selected .PFM files or place them back in the left-hand "available" list box by selecting filenames in the same manner in the right-hand "selected" list box.

When you click the OK button, the .PFM files that appear in the right-hand list box are placed in the .PCM file. To create the .PCM file, you must select at least one .PFM file and have a non-empty title string.

When the editor creates the .PCM file , it also creates a text file in the same directory with the same name and the extension .INI. On the first line of this .INI file is the title of the cartridge; the list of .PFM files follows on subsequent lines, one filename per line. Whenever the PCM dialog box is chosen, the PFM Editor looks for the .INI file and uses the stored data to initialize the title string and the list of selected .PFM files.

Notice, however, that editing a .PFM file will not cause the .PCM file to be updated. The PFM Editor cannot directly edit .PCM files. Therefore, if you make a change in a .PFM file, you must rebuild the .PCM file.

4.7 The PFM Editor Error Messages

The following error messages can appear when using the PFM Editor. A brief explanation of why it appeared is given after each message.

Break character not in character set
The break character must be between the first and last character in the character set, inclusive.

Can't find *filename*, ignored
The .INI file for a .PCM file contained a reference to a .PFM file that is not in the same directory, or that does not have the .PFM extension.

Can't open *filename*, ignored
The given .PFM file was in the current directory when the list box of available .PFM files was created. However, when the attempt to write the .PCM file out was made, the .PFM file could not be opened to read it.

Can't write the PCM file
An error occurred while writing to the .PCM file or to the associated .INI file.

Changing the character range will invalidate the width table
If you enter a width table and, then, change the first and last characters, the width table will no longer be valid. This really shouldn't occur if you have a specific font in mind when you create a .PFM file.

Couldn't save file *filename*
The file specified could not be saved to disk due to an error opening or writing the file, or to attempting to write over a Read-Only file, or to running out of disk space.

Default character not in character set
The default character must be between the first and last character in the character set, inclusive.

Driver information not specified
You attempted to save a file that does not contain PCL driver information.

Error creating the initialization file for *filename*
The .INI file corresponding to the given file could not be opened.

Error creating the PCM file *filename*
The .PCM file could not be opened due to a bad filename (such as a non-existent directory), or to a pre-existing Read-Only file of the same name, or to being out of disk space.

filename contains the wrong driver information version
This is a warning indicating that while the .PFM file appears to be in the correct format, the driver specific information contains a version number that this PFM Editor does not support.

filename contains the wrong header version number
This is a warning indicating that the PFM header has a version number that this PFM Editor does not recognize.

filename does not contain driver information
The .PFM file does not contain a PCL driver structure.

filename does not contain extended text metrics
This is a warning that the .PFM file being read into the PFM Editor does not contain extended text metrics.

filename has been modified. Save before continuing?
Allows you to save a file or abort an operation before doing something that will abandon a modified .PFM file, such as choosing New, Open..., or Exit.

filename is not a valid file name
The filename given is invalid because it contains illegal characters or syntax errors.

filename is not a valid PFM file
The specified file does not contain a .PFM file, or the .PFM file is either corrupted or from the wrong version.

Proportional font requires a width table
You attempted to save a .PFM file that specifies a proportional font but does not contain a width table.

Unrecognized device *devicename*
The PFM contains a device name that the PFM Editor does not recognize. The only name recognized currently is "PCL/HP Laserjet."

Width table will need to be reentered
This is another warning that changing your character range invalidates your width table.

Write over original *filename*?
A file with the same name as the filename you specified in the Save As... dialog already exists. Click **Yes** to write over the old file or No to choose a new name.

CHAPTER 5

.PFM FILES FOR POSTSCRIPT PRINTERS

CONTENTS

5 .PFM FILES FOR POSTSCRIPT PRINTERS

This chapter discusses the format of .PFM files designed specifically for PostScript printers. We recommend you read Chapter 3, "Printer Font Metric (PFM) Files," first for a description of the general format of .PFM files.

5.1 Format of the PostScript .PFM File

The PostScript .PFM file does not contain a width table. Because PostScript fonts are scalable, the widths are contained in the extent table. The structures in the file are organized as follows:

1. PFM header structure (PFMHEADER)
2. PFM extension structure (PFMEXTENSION), must contain a non-NULL pointer to *dfExtMetricsOffset*, *dfExtentTable* and *dfDriverInfo*
3. Driver name pointed to by *dfDevice*
4. Windows font name pointed to by *dfFace*
5. Extended Text Metrics structure (EXTTEXTMETRICS) pointed to by *dfExtMetricsOffset*
6. Extent table pointed to by *dfExtentTable*
7. PostScript font name pointed to by *dfDriverInfo*

The file may also contain:

8. Pair-kern table pointed to by *dfPairKernTable*
9. Track-kern table pointed to by *dfTrackKernTable*

This is the recommended organization of the file. The PFM header must be the first structure in the file and the PFM extension must be the second structure. The remainder of the structures may appear in any order. Their locations are indicated by the offsets in the PFM header and PFM extension structures.

The Windows PostScript driver assumes all PostScript fonts are scalable fonts, so it ignores the *dfPoints* and *dfPixHeight* fields in the PFM header. The fields *dfAvgWidth* and *dfMaxWidth* are in units of 1000 units/em.

Although the PostScript naming convention includes the attributes of the font (bold, italic) in the font name, the attributes should be stripped from the font name and represented in the *dfWeight* and *dfItalic* fields in the PFM header.

The extent table is an array of words containing the unscaled widths of the characters and assuming 1000 units/em. The range of the table should be from *dfFirstChar* to *dfLastChar*. The size of the table should be *dfLastChar* - *dfFirstChar* + 1 word.

Pair-kern values should be in the same 1000 units/em measurement as the extents. As of this writing, we do not know of any application that uses the track-kern table.

5.2 Extended Text Metrics Structure

The definition of *etmMasterHeight* and *etmMasterUnits* is provided in detail in Chapter 3, "Printer Font Metrics (PFM) Files." The PostScript driver assumes these values for each font:

dfVertRes = 300
etmMasterHeight = 300
etmMasterUnits = 1000

In other words, the driver assumes all fonts use Adobe's standard 1000 units/em method for describing a font. You must build the extent table based upon 1000 units/em to be consistent with this restriction in the driver.

The driver also assumes that the font may be scaled to any point size the application requests. We recommend that the true scaling range of the font be indicated in *etmMinScale* and *etmMaxScale* (in device units, at 300dpi). Even though the driver currently ignores these fields, it may use them in the future.

Because the Windows PostScript driver assumes all PostScript fonts are scalable fonts, it ignores the *etmPointSize* field. Please notice that *etmSize* is not the point size but, rather, the size (i.e., the number of bytes) of the extended text metrics structure.

As of this writing, we do not know of any application that uses the fields in the extended text metrics structure except for *etmKernPairs*. If your font contains kern pairs, you must fill in the extended text metric structure to indicate the number of kern pairs. Do not leave the other fields blank; fill them in with reasonable values in the event an application does use them.

5.3 Driver-Specific Structure

The driver-specific data structure pointed to by *dfDriverInfo* is a null-terminated string containing the PostScript name for the font. There are really two names for the font:

1. The Windows name for the font, such as "Tms Rmn," which appears in the font list in the application's font dialog box.

2. The PostScript name for the font, such as "Times-Roman," which the driver sends to the printer to select the font.

Both strings must be null-terminated. The Windows name for the font is pointed to by *dfFace* and the PostScript name for the font is pointed to by *dfDriverInfo*.

CHAPTER 6

.PFM FILES FOR PCL PRINTERS

CONTENTS

6 .PFM FILES FOR PCL PRINTERS

This chapter discusses the format of .PFM files designed specifically for PCL printers. If you have not already done so, we recommend you read Chapter 3, "Printer Font Metrics (PFM) Files," first for a general description of .PFM files.

6.1 Format of the PCL .PFM File

The structures in the PCL PFM file are organized as follows:

1. PFM header structure (PFMHEADER)
2. For variable-width fonts, an array of character widths from *dfFirstChar* to *dfLastChar*
3. PFM extension structure (PFMEXTENSION)
4. Driver name pointed to by *dfDevice*
5. Windows font name pointed to by *dfFace*

The file may also contain:

6. Extended Text Metrics structure (EXTTEXTMETRICS) pointed to by *dfExtMetricsOffset*
7. Driver-specific structure pointed to by *dfDriverInfo*
8. Pair-kern table pointed to by *dfPairKernTable*
9. Track-kern table pointed to by *dfTrackKernTable*

This is the recommended organization of the file. The PFM header must be the first structure in the file followed by the width table and the PFM extension structure. The remainder of the structures may appear in any order. Their locations are indicated by the offsets in the PFM header and PFM extension structures.

The width table is present if *dfPixWidth* is non-zero, which indicates a variable-width font. (Otherwise, for fixed-width fonts, the width of all the characters in the font equals the value in *dfPixWidth*.) The width table is an array of words containing the widths in device units of characters in the range from *dfFirstChar* to *dfLastChar*. The width of a character *c* can be located using the formula:

width = *WidthTable*[*c* - *dfFirstChar*]

The size of the table is *dfLastChar* - *dfFirstChar* + 2 words. The last word is not used and is set to NULL; it should be present for compatability with the Windows screen-font file format.

6.2 Extended Text Metrics Structure

As of this writing, we do not know of any application that uses the fields in the extended text metrics structure except for *etmKernPairs*. If your font contains kern pairs, you must fill in the extended text metric structure to indicate the number of kern pairs. Do not leave the other fields blank; fill them in with reasonable values in the event an application does use them.

In the future, there will be .PFM files that describe scalable PCL fonts. To guarantee that your raster fonts are never interpreted as scalable fonts, make sure that the fields for scaling information in the extended text metrics structure indicate a non-scaling font:

etmMasterHeight = *etmMasterUnits* = *etmMinScale* = *etmMaxScale* = *dfPixHeight*

See Section 5.4, "Scalable PCL Fonts," for information on how to write a .PFM file for scalable PCL fonts.

6.3 Driver-Specific Structure

The SYMBOLSET field is defined as:

```
typedef enum{
        epsymUserDefined,               /* Enumerated type SYMBOLSET */
        epsymRoman8,
        epsymKana8,
        epsymMath8,
        epsymUSASCII,
        epsymLineDraw,
        epsymMathSymbols,
        epsymUSLegal,
        epsymRomanExt,
        epsymISO_DenNor,
        epsymISO_UK,
        epsymISO_France,
        epsymISO_German,
        epsymISO_Italy,
        epsymISO_SwedFin,
        epsymISO_Spain,
        epsymGENERIC7,
        epsymGENERIC8,
        epsymECMA94
        }SYMBOLSET;
```

The TRANSTABLE field is defined as:

```
typedef struct{
        /* type TRANSTABLE */
        SYMBOLSET symbolSet;            /* kind of translation table */
        DWORD offset;                   /* location of user-defined table */
        WORD len;                       /* length (in bytes) of table */
        BYTE firstchar,lastchar;        /* table range */
        }TRANSTABLE
```

The driver-specific data structure pointed to by *dfDriverInfo* is:

```
typedef struct{
        WORD epSize;                    /* size of this data structure */
        WORD epVersion;                 /* number indicating version of struct */
        DWORD epMemUsage;               /* amt of memory font takes up in printer */
        DWORD epEscape;                 /* pointer to escape that selects the font */
        TRANSTABLE xtbl;                /* character set translation info */
        }DRIVERINFO;
```

The fields in the structure are defined as follows:

Field	Definition
epSize	The size of this structure (number of bytes)
epVersion	The version of this structure, currently 1
epMemUsage	The amount of *printer* memory, in bytes, that this font uses
epEscape	The byte offset from the beginning of the file to an escape string invoking the font. A future release of the driver will parse this string for the reserved word "#HEIGHT" and will replace every occurance of it with the height (that is, *dfPixHeight*) of the font.
xtbl.xtSymbolSet	Symbol set ID, such as *epsymRoman8* or *epsymECMA94*
xtbl.xtOffset	The byte offset from beginning of the file to a custom translation table
xtbl.xtLen	The size of the custom translation table
xtbl.xtFirstChar	The first character translated in the table
xtbl.xtLastChar	The last character translated

The purpose of the translation table is to allow the driver to translate the font from the character set indicated in the *dfCharSet* field into the printer-specific character set. When *xtbl.xtSymbolSet* equals *epsymUserDefined*, the driver would use the custom translation table pointed to by *xtbl.xtOffset*. However, the Windows PCL driver currently ignores all fields in the TRANSTABLE structure except *xtbl.xtSymbolSet*. In other words, even if you put in a custom translation table for future support, the current driver will ignore it.

The Windows PCL driver supports five possible character translations. It determines which internal translation table to use based upon the value of *xtbl.xtSymbolSet*. The tables are as follows:

Symbol Set	Translation Table
epsymRoman8	Roman8_Trans
epsymUSASCII	USASCII_Trans
epsymECMA94	ECMA94_Trans
epsymGENERIC8	GENERIC8_Trans
epsymGENERIC7	GENERIC7_Trans

The translation tables are stored in *trans.h*. For *epsymRoman8*, *epsymUSASCII*, and *epsymECMA94*, the driver attempts to derive Windows ANSI from the symbol set. For *epsymGENERIC8* and *epsymGENERIC7*, the driver simply lets an 8- or 7-bit symbol set pass through to the printer unchanged.

The driver assumes the width table in the .PFM file contains the widths of the characters *AFTER* translation. If you set *epsymRoman8*, *epsymUSASCII*, or *epsymECMA94* for *xtbl.xtSymbolSet*, you must use the appropriate translation table in *trans.h* to do an inverse translation when building the width table.

A portion of the translation table for *epsymRoman8* looks like this:

```
#define HP_DF_CH ((BYTE) 0x7F)

unsigned char Roman8_Trans[] = {
        HP_DF_CH,       NULL,   /*  80  */
        HP_DF_CH,       NULL,   /*  81  */
        ...
        'Y' ,   0xa8,   /*  dd  */
        0xf0,   NULL,   /*  de  */
        0xde,   NULL,   /*  df  */
        0xc8,   NULL,   /*  e0  */
        ...
        0xef,   NULL  };        /*  ff  */
```

The table translates characters in the range from 128 to 255. The driver uses the character it receives from the application to index into the translation table. It replaces the character with the first entry in the table. If the second entry is non-NULL, it overstrikes the first character with the second character.

For example, when the driver detects character hex 0xDD,(i.e., the Y-acute (Y) in the text stream, the driver will output a capital "Y" overstruck by the acute accent. When the overstrike character is present, the driver guarantees the width of the character pair equals the width of the first character.

If the driver-specific data structure is not present or *xtbl.xtSymbolSet* equals a symbol set other than *epsymRoman8*, *epsymUSASCII*, *epsymECMA94*, *epsymGENERIC8*, or *epsymGENERIC7*, the driver will default to *epsymGENERIC8* translation if *dfLastChar* is greater than 127 (an 8-bit font). Otherwise, it will use the *epsymGENERIC7* translation.

6.4 Kerning Tables

The pair-kern table follows the format described in Chapter 3, "Printer Font Metrics (PFM) Files." The kern amounts are in the same units as the character widths.

As of this writing, we do not know of any application that uses the track-kern table.

6.5 Scalable PCL Fonts

Currently, the Windows PCL driver supports only non-scaling PCL fonts. In the future, however, it may support scalable fonts if PCL printers with scaling fonts become available. The driver will detect scalable fonts by examining the *etmMinScale* and *etmMaxScale* fields in the extended text metrics structure. If they are equal, it will assume a non-scaling font. The font vendor should provide a .PFM file that follows this format:

1. PFM header structure (PFMHEADER). The *dfPixHeight* field must contain the height of a default font size (same as 12 points).
2. For variable-width fonts, an array of character widths from *dfFirstChar* to *dfLastChar* for the default point size (*dfPixHeight*)
3. PFM extension structure (PFMEXTENSION)
4. Driver name pointed to by *dfDevice*
5. Windows font name pointed to by *dfFace*
6. Extended Text Metrics structure (EXTTEXTMETRICS) pointed to by *dfExtMetricsOffset*
7. Extent table pointed to by *dfExtentTable*
8. Device-specific data structure pointed to by *dfDriverInfo*. The *epEscape* field must be non-NULL (that is, a printer escape must be provided).

Optionally, pair-kern and track-kern tables may be provided. The default font size and width table should be provided for consistency with the existing driver. A future release of the driver will check for the difference between *etmMinScale* and *etmMaxScale*. If they differ and *dfExtMetricsOffset* is non-NULL, the driver will assume a scalable font.

An extent table must be supplied for scalable fonts. The extent table is an array of words containing the unscaled widths of the characters. The range of the table should be from *dfFirstChar* to *dfLastChar*. The size of the table should be *dfLastChar* - *dfFirstChar* + 1 word.

The driver will scale the characters using the formulas described in Section 3.3.4, "Font Scaling: etmMasterHeight and etmMasterUnits." The driver will not support the ENABLERELATIVEWIDTHS escape, as this would be difficult to support with scaling and non-scaling fonts intermixed (scaling fonts use font units; non-scaling fonts use device units).

The driver will assume *dfVertRes* equals 300dpi. Remember that *etmMasterHeight*, *etmMinScale*, and *etmMaxScale* must be expressed in device units.

If the target printer does not force its widths to 300dpi units (*etmMasterUnits* does not equal *etmMasterHeight*), the driver will attempt to correct for roundoff error between the printer's units and the driver's imposed 300dpi units. The driver will correct for the error by maintaining a running roundoff value during output of a single line of text.

You must provide the driver-specific data structure. As described earlier, *xtbl.xtSymbolSet* must be equal to *epsymRoman8*, *epsymUSASCII*, *epsymECMA94*, *epsymGENERIC8*, or *epsymGENERIC7*. Remember to use an inverse translation of the tables provided in *trans.h* to build the extent table if you select *epsymRoman8*, *epsymUSASCII*, or *epsymECMA94*.

You must also provide an escape string pointed to by *epEscape* in the driver-specific data structure. The driver will parse this string for the reserved word "#HEIGHT" and will replace every occurance of it with the desired height of the font. The driver will send this escape to the printer to select the font.

If pair or track kerns exist, they should use the same units as the character widths in the extent table.