AFM Files:
An Interchange Format for
POSTSCRIPT® Font Metrics

Adobe Systems, Inc.

Draft of 09 March 87 15:22

This document describes a standard interchange format for communicating font metric information to people and programs. The format is ASCII encoded (for both human and machine readability), machine independent, and extensible. Files in this format will be known as Adobe Font Metrics (AFM) files and will be available for all of Adobe Systems's POSTSCRIPT[1] fonts.

- The format is ASCII encoded and human readable. The advantages of this approach are obvious from our success with POSTSCRIPT itself. Applications may choose to ''compile'' the metrics information into another form (e.g., TEX[2] TFM or PL files, fonts.widths format, troff vfont format, etc.).
- The format is extensible. Virtually all existing metrics file formats are overconstrained. They suffer from the fact that they become quickly outdated, yet still in use; thus limiting the capabilities of applications that use them. AFM format allows new non-standard fields to be added with relative ease.
- The format is readily parsed, with the ability for applications to quickly skip over items that are not of interest.
- Almost the entire AFM format (except for kerning and composite character data) can be generated by POSTSCRIPT from the information embodied within a POSTSCRIPT font.

Each AFM file contains the information for one POSTSCRIPT printer font. The file begins with global information pertaining to the font as a whole, followed by sections with character metrics, kerning data (optional) and composite character data (optional). The file format is line-oriented, each line beginning with a property (key) name, followed by the values for that property. Keys and values are separated by one or more white space characters (space or tab).

The format will be:

Key *value value ...*

Key names are case-sensitive. All keys beginning with a capital letter are reserved for use by Adobe Systems; user-defined non-standard entries should begin with a lowercase letter. The Adobe Systems standard keys are detailed below, but other keys are allowed and should be ignored by programs not recognizing them.

Values will be one of the following: string, name, number, integer or boolean. Strings are terminated by the end of line. Names are similar to strings except that they may not contain any white space characters; they are terminated by white space characters or by a special termination character (see section on ''Character Metrics'' below). Numbers, integers and booleans are separated by white space characters. A number can be either a real number or an integer. A boolean is either the value **true** or **false**.

An AFM file begins with the line:

StartFontMetrics *version*

The version described here is 2.0. The last line of the file will be:

EndFontMetrics

Comments may be inserted at any point in the AFM file. The following notation is used:

Comment *string* Comment strings may be ignored.

[1] POSTSCRIPT is a registered trademark of Adobe Systems, Inc.

[2] TEX is a trademark of the American Mathematical Society.

Global Font Information

The following global font keys are the same as those in the top level or **FontInfo** subdictionary of an Adobe Systems POSTSCRIPT font dictionary. Their meanings are described in the chapter on fonts in the *POSTSCRIPT Language Reference Manual*. Please note that although some of the keys in the POSTSCRIPT **FontInfo** subdictionary begin with a lowercase letter (e.g., **isFixedPitch**, **version**) all keys listed here begin with uppercase letters to distinguish them as keys reserved for use by Adobe Systems. All numeric values are in the character coordinate system (1000 units per em).

FontName *string* — Name of the font as presented to the POSTSCRIPT **findfont** operator. *Example:* Garamond-Light.

FullName *string* — The full text name of the font. *Example:* ITC Garamond Light.[3]

FamilyName *string* — The name of the ‘‘font family’’ to which the font belongs. *Example: ITC Garamond*.

Weight *string* — Weight of the font. *Example:* Roman, Bold, Light, etc.

ItalicAngle *number* — Angle (in degrees counter-clockwise from the vertical) of the dominant vertical strokes of the font. *Example:* -12.

IsFixedPitch *boolean* — If *boolean* is **true**, this indicates that the font is a fixed pitch (monospaced) font.

FontBBox *llx lly urx ury* — Four *numbers* giving the lower left corner and the upper right corner of the font bounding box. *Note:* the bounding box given here is that of the flattened paths, not the Bezier curve descriptions.

UnderlinePosition *number* — Distance from the baseline for centering underlining strokes.

UnderlineThickness *number* — This is the stroke width for underlining.

Version *string* — Font version identifier.

Notice *string* — Font name trademark or copyright notice.

The following keys are also standard:

EncodingScheme *string* — String indicating the default encoding vector for this font. The most common one is *AdobeStandardEncoding*. Special fonts may simply state *FontSpecific*.

CapHeight *number* — Top of capital H.

XHeight *number* — Top of lower case x.

Ascender *number* — Top of lower case d.

Descender *number* — Bottom of lower case p.

[3] ITC Garamond is a registered trademark of International Typeface Corporation.

Character Metrics

The set of metrics for individual characters is surrounded by the lines **StartCharMetrics** and **EndCharMetrics**. The **StartCharMetrics** keyword is followed by an integer indicating the number of individual characters to expect. The format of the line actually is **StartCharMetrics** *integer*. This integer is provided so that a data consistency check can be performed by the program parsing this data.

Each character's metrics consists of a list of keys and values separated by semicolons; the metrics for a given character will always be contained in one line. The characters are sorted by numerically ascending character code. Unencoded characters follow the encoded characters and are identified by character codes of −1. Standard keys are:

C *integer*	Decimal value of default POSTSCRIPT character code (−1 if unencoded).
WX *number*	Character width in *x* (*y* is 0).
W $number_x$ $number_y$	Character width vector *(x,y)*
N *name*	POSTSCRIPT character name.
B *llx lly urx ury*	Character bounding box where *llx, lly, urx,* and *ury* are all *numbers*.
L *successor ligature*	Ligature sequence where *successor* and *ligature* are both *names*. The current character may join with the character named *successor* to form the character named *ligature*. Note that characters may have more than one such entry.

Example: A character metric data line might look like this:

C 102; WX 333; N f; B 21 0 382 685; L i fi; L l fl;

Kerning Data

The kerning data section is optional; it may or may not be present for a given font. The section is surrounded by the lines **StartKernData** and **EndKernData**. Kerning data is supplied in two forms: track kerning and pair-wise kerning. Track kerning is applied to all characters uniformly whereas pair-wise kerning is applied to specific character pairs. Track kerning and pair-wise kerning may be used independently or together (i.e., it is possible to apply track kerning to a line of text and then to apply pair-wise kerning on top of that). The two forms of kerning data are treated as subsections within the kerning data section and both sections need not be present.

The track kerning data is surrounded by the lines:

StartTrackKern *integer*	Where *integer* indicates how many different tracks are present.
EndTrackKern	Signals the end of the track kerning data.

Normally track kerning is provided in different *degrees* of tightness. Within a track (a degree of tightness), the amount to decrease (or possibly increase) the amount of space between characters increases (or possibly decreases) with the point size of the font (e.g., for tight track kerning, the amount to decrease the space between characters at 6 point might be 0.1 points and at 72 point it might be 3.78 points).

The data itself begins with the key **TrackKern** and is followed by the track kerning information:

TrackKern *degree min-pt-size min-kern-amt max-pt-size max-kern-amt*

The *degree* is an integer where increasingly negative degrees represent tighter track kerning and increasingly positive degrees represent looser track kerning. *min-pt-size*, *min-kern-amt*, *max-pt-size* and *max-kern-amt* are all *numbers*. Since the track kerning is a linear function, the minimum and maximum cut-off values (point sizes) are provided along with the amount to track kern by at the point size. The kerning amounts are given relative to the point size. From those 4 values, the track kerning function can be derived. The track kerning function is a linear function. The equation for the line can be determined from the data provided and, therefore, the track kerning values for any point size can be determined. The

track kerning values for any point size below/above the minimum/maximum point size are constant (the minimum kerning amount/maximum kerning amount).
In general the track kerning function is as follows:

Let x = current point size
Let k_1 = max-kern-amt, k_0 = min-kern-amt
Let p_1 = max-pt-size, p_0 = min-pt-size

$$
\begin{aligned}
f(x) &= k_0 && \text{for } x < p_0 \\
f(x) &= \frac{(k_1 - k_0)}{(p_1 - p_0)} * x + \frac{(k_0 * p_1 - k_1 * p_0)}{(p_1 - p_0)} && \text{for } p_0 \le x \le p_1 \\
f(x) &= k_1 && \text{for } x > p_1
\end{aligned}
$$

Example:

StartTrackKern 3
Comment Light kerning
TrackKern -1 14 0 72 -1.89
Comment Medium kerning
TrackKern -2 8 0 72 -3.2
Comment Tight kerning
TrackKern -3 6 -.1 72 -3.78

Below is a sample of text printed using these track kerning values.

6 pt

no kerning An illustration of how track kerning works.
light kerning An illustration of how track kerning works.
medium kerning An illustration of how track kerning works.
tight kerning An illustration of how track kerning works.

12 pt

no kerning An illustration of how track kerning works.
light kerning An illustration of how track kerning works.
medium kerning An illustration of how track kerning works.
tight kerning An illustration of how track kerning works.

18 pt

no kerning An illustration of how track kerning works.
light kerning An illustration of how track kerning works.
medium kerning An illustration of how track kerning works.
tight kerning An illustration of how track kerning works.

Figure 1: Track kerning

The pair-wise kerning data is surrounded by the lines:

StartKernPairs *integer* Where *integer* indicates how many pairs to expect.

EndKernPairs Signals the end of the pair-wise kerning data.

There will be one kerning pair per line. Each line will begin with a keyword of the form **KP** or **KPX**.

KP $name_1$ $name_2$ $number_x$ $number_y$

Name of the first character in the kerning pair followed by the name of the second character followed by the kerning vector specified as an *(x,y)* pair. The kerning vector is the amount to move the second character by relative to the first character to position it properly. The kerning vector is specified in the character coordinate system. In order to use this vector it is necessary to transform it into user space and scale it by the point size in use. The best way to do this is to use the **FontMatrix** entry in the current font dictionary.

KPX $name_1$ $name_2$ $number_x$

Name of the first character in the kerning pair followed by the name of the second character followed by the kerning amount in the x direction (y is zero). The kerning amount is specified in the units of the character coordinate system.

A character pair kerning line might look like this:

KPX V A -129

Below is an example of pair-wise kerning applied to 100 point characters:

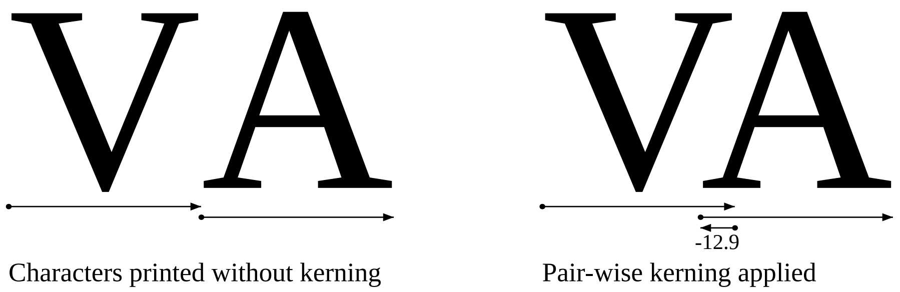

Figure 2: Pair-wise kerning

Composite Character Data

The composite character data section is also optional. Composite characters are new characters that are made up of characters already existing in the font, such as accented characters. Character metric information for composite characters is found in the Character Metrics section of the AFM file. Although most POSTSCRIPT fonts available from Adobe Systems include a rather extensive set of composite characters, some applications may wish to generate their own. This section provides the data necessary for accurate positioning of the individual pieces.

If present, the composite character data is surrounded by the lines **StartComposites** and **EndComposites**. The **StartComposites** keyword is followed by an integer indicating how many composite characters to expect. Again, all units are in the character coordinate system (1000 units per em).

The data for each composite character is represented as a list of keys and values separated by semicolons. Each composite character gets one line of description. The standard keys are:

CC *name integer* — The composite character name followed by the number of parts that make up the composite.

PCC *name deltax deltay* — One of the parts of the composite character. The character name is given followed by the x and y displacement from the origin (all units are in the character coordinate system).

A composite character line might look like this:

CC Aacute 2; PCC A 0 0; PCC acute 194 214;

Below is an example of how the individual pieces of a composite character are combined to form the composite:

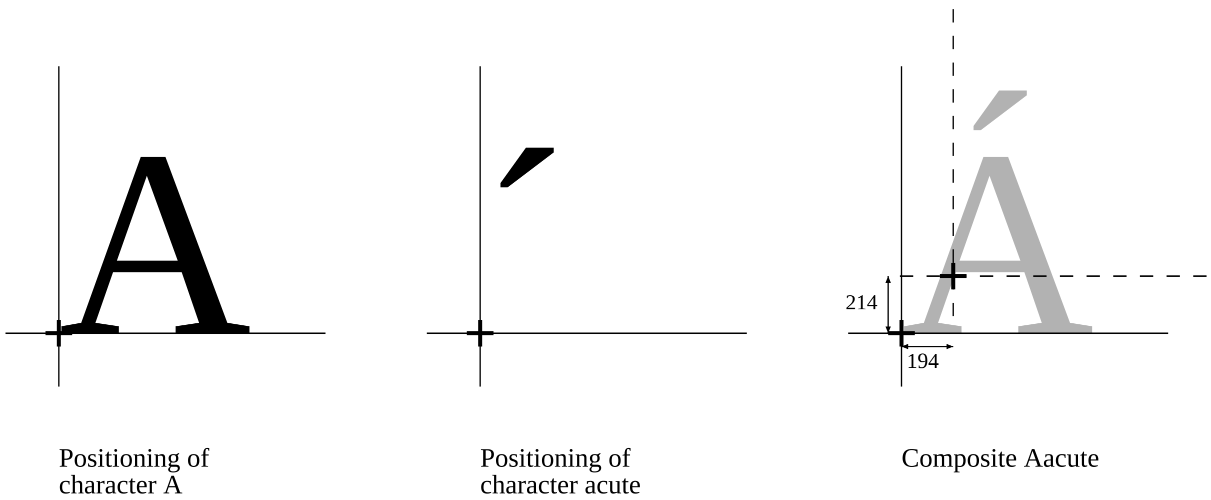

Figure 3: Example of positioning for a composite character

The following is a complete example of an AFM file for Times[4] Roman.

StartFontMetrics 2.0
Comment Copyright (c) 1984 Adobe Systems Incorporated. All Rights Reserved.
FontName Times-Roman
FullName Times Roman
FamilyName Times
Weight Medium
ItalicAngle 0.0
IsFixedPitch false
UnderlinePosition -98
UnderlineThickness 54
Version 001.000
Notice Times is a trademark of Allied Corporation.
EncodingScheme AdobeStandardEncoding
FontBBox -167 -252 1004 904
CapHeight 673
XHeight 445
Descender -219
Ascender 686
StartCharMetrics 210
C 32 ; WX 250 ; N space ; B 0 0 0 0 ;
C 33 ; WX 333 ; N exclam ; B 128 -17 240 673 ;
C 34 ; WX 408 ; N quotedbl ; B 46 445 313 685 ;
C 35 ; WX 500 ; N numbersign ; B 20 -17 481 673 ;
C 36 ; WX 500 ; N dollar ; B 45 -92 456 726 ;
C 37 ; WX 833 ; N percent ; B 63 -36 771 655 ;
. . . -- *lines omitted for brevity* --
C 101 ; WX 444 ; N e ; B 24 -17 416 469 ;
C 102 ; WX 333 ; N f ; B 21 0 382 685 ; L i fi ; L l fl ;
C 103 ; WX 500 ; N g ; B 24 -220 469 468 ;
C 104 ; WX 500 ; N h ; B 11 0 487 686 ;
C 105 ; WX 278 ; N i ; B 25 0 255 685 ;
C 106 ; WX 278 ; N j ; B -56 -217 205 685 ;
C 107 ; WX 500 ; N k ; B 7 0 496 686 ;
. . . -- *lines omitted for brevity* --
C 248 ; WX 278 ; N lslash ; B 0 0 283 685 ;
C 249 ; WX 500 ; N oslash ; B 30 -104 469 566 ;
C 250 ; WX 722 ; N oe ; B 30 -10 684 462 ;
C 251 ; WX 500 ; N germandbls ; B 13 0 468 686 ;
C -1 ; WX 722 ; N Aacute ; B 22 0 702 873 ;
C -1 ; WX 722 ; N Acircumflex ; B 22 0 702 875 ;
C -1 ; WX 722 ; N Adieresis ; B 22 0 702 819 ;
. . . -- *lines omitted for brevity* --
EndCharMetrics
StartKernData
StartTrackKern 3
TrackKern -1 14 0 72 -1.89
TrackKern -2 8 0 72 -3.2
TrackKern -3 6 -.1 72 -3.78
EndTrackKern

---

[4]Times is a trademark of Allied Corporation.

StartKernPairs 2
KPX V A -129
KPX A Y -92
EndKernPairs
EndKernData
StartComposites 1
CC Aacute 2; PCC A 0 0; PCC acute 194 214;
EndComposites
EndFontMetrics

Table of Contents

List of Figures