The Type 42 Font Format Specification

Adobe Developer Support

Technical Note # 5012

31 July 1998
Contents

The Type 42 Font
Format Specification 1

1 Introduction 1

2 The Type 42 Font Format 2
   Type 42 Font Comment Lines 3
   The Type 42 Font Dictionary 3
   Implications of The Glyph Coordinate System 6

3 Identifying Interpreters with TrueType Rasterizers 6

4 Conversion Issues 7
   The FontInfo Dictionary 7
   The sfnts Array 7
   Generating The CharStrings Dictionary 9
   Generating the Encoding Vector 9
   Glyph Mapping and Metrics Access 10
   Generating Unique Identifiers 11
   Required TrueType Tables 12

5 CIDFontType 2 CID Fonts 12
   Complete Font Downloading 13
   Incremental Font Downloading 14
   CMap Resources for CIDFontType 2 CID Fonts 15
   Changes to CIDMap 15
   GlyphDirectory 16
   Vertical Writing Mode 18
   MetricsCount 18

6 Known Bugs 20

7 Example Type 42 single-byte font program 20

Index 23
1 Introduction

This document describes the PostScript® Type 42 font format which can be used to download TrueType® fonts to PostScript printers (or PostScript compatible printers) that contain a TrueType rasterizer. This method yields better print quality than can be achieved by converting a TrueType font to a Type 1 or Type 3 font. Performance may also be improved for large Chinese, Japanese, or Korean (CJK) fonts when the print driver takes advantage of glyph subsetting and incremental downloading.

A Type 42 font dictionary contains the TrueType font embedded as a string value for the sfnts keyword. For single-byte Roman TrueType fonts, the sfnts array typically contains the complete TrueType font. This was first supported in PostScript version 2010 for printers that contain the optional TrueType rasterizer. Other entries in the Type 42 font dictionary permit the PostScript interpreter to handle the font in a manner similar to a Type 1 font, and to make the TrueType font data available to the TrueType rasterizer.

This version of this document, dated June 1998, adds a description of how to convert multi-byte TrueType fonts into CIDFontType 2 CID fonts. This takes advantage of the CID format’s ability to encode the large number of glyphs needed for most CJK fonts. Both permanent downloading to a printer’s hard disk, and font embedding for a print job are explained.

Support for multi-byte CJK TrueType fonts was added in version 2015 of the PostScript interpreter by providing a mechanism for subsetting and incremental downloading of large fonts. Recent updates for version 3011 support further optimizations to improve printing performance.

This document only describes the format of a Type 42 font program and how it may be created from a TrueType font. The TrueType specification from Apple Computer is available at the following URL:


Microsoft’s TrueType specification is available at the following URL:
Additional information on Type 42 fonts can be found in: The PostScript Language Reference Manual Supplements for versions 2015, 3010, and 3011. Also, see Adobe Technical Note #5213, “PostScript Language Extensions for CID-Keyed Fonts (PostScript Software Version 2015).”

2 The Type 42 Font Format

The TrueType font format was originally developed by Apple Computer, and is the native font format used by the Macintosh and Windows operating environments. Until recently, Macintosh print drivers would download either the TrueType font and the TrueType rasterizer to interpreters with 680X0-class controllers, or convert the font into a PostScript Type 1 (unhinted) font program. Windows print drivers would either convert to Type 1 or Type 3, or rasterize the TrueType font on the host, and download a bitmap Type 3 or Type 32 font. All of these methods have disadvantages: the TrueType rasterizer is large; conversion to Type 1 format loses the hint information and the translation of TrueType quadratic to PostScript Bézier curves cannot be exact; and bitmap fonts cannot be scaled for multiple resolutions retain the necessary quality.

Many PostScript interpreters, beginning with version 2013, now include a TrueType rasterizer; they can be identified from the PostScript Printer Description (PPD) file (see section 3). For a TrueType font to be recognized by a PostScript interpreter, it must be enclosed in a PostScript font dictionary with \texttt{FontType 42}, or as a CID font with \texttt{CIDFontType 2} and \texttt{FontType 42}.

\texttt{FontType 42} and \texttt{CIDFontType 2} are standard part of LanguageLevel 3; all version 3010 and greater interpreters support them. However, the version 3011 extensions are not part of LanguageLevel 3.

\textit{Note} Although the keyword name \texttt{sfnts} is derived from the Macintosh resource type used for TrueType fonts, TrueType fonts from the Windows environment can be converted in an identical manner.

A Type 42 font is a base font and shares all the properties of base fonts as documented in the \textit{PostScript Language Reference Manual, second edition}. In particular, presence of an unique identifier such as an \texttt{XUID} facilitates bitmap caching for a Type 42 font just as it does for any other type of base font. When Type 42 fonts are permanently downloaded to a hard disk connected to a PostScript printer, only the \texttt{charstrings} actually referenced in the print job will be read into VM, thus saving memory (see section 4.2). Also, glyphs can be modified or added to the Type 42 font by using user-defined PostScript language procedures (See section 5.6.3 in \textit{The PostScript Language Reference Manual, second edition}).
2.1 Type 42 Font Comment Lines

The first line of a Type 42 font program shall be:

```
%!PS-TrueTypeFont-<TTVersion>-<MfrRevision>
```

where `<TTVersion>` is the TrueType version number of the font (specified in the header); `<MfrRevision>` is the font manufacturer’s revision number of the font. This line helps downloaders to easily identify TrueType Type 42 fonts on the printer’s hard disk.

The first portion of the comment line: `%!PS-TrueTypeFont`, is required for disk-based Type 42 fonts on any interpreter that supports Type 42. If the font has the required portion of the comment line and it conforms to the requirements listed in section 4.2, the embedded TrueType glyph data will be accessed from disk on demand rather than reading the entire font into VM.

Another useful and recommended comment line specifies the VM usage:

```
%%VMusage: <MinMemory> <MaxMemory>
```

where `<MinMemory>` and `<MaxMemory>` specify the minimum memory needed for the font and how much is needed if the font is downloaded first (these numbers are not necessarily the same; see *The Adobe Type 1 Font Format, version 1.1*, Addison Wesley, 1990). This comment is not used by the PostScript interpreter, but is useful for application programs. The PostScript operator `resourcestatus` can be used to obtain the VM requirements for a font resource.

The values for the `VMusage` comment can be derived from a TrueType font which contains a `post` table (which contains information useful for PostScript printing). For downloading purposes, the size of the font can be used as an estimate for the values for `VMusage`. However, a properly constructed Type 42 font on a printer’s hard disk will generally require only a percentage of the VM required for the downloadable version of the font.

2.2 The Type 42 Font Dictionary

Table 1 lists entries common to all types of font dictionaries. Table 2 lists additional key-value pairs that are meaningful in all `base` fonts. Table 3 lists additional key-value pairs that are meaningful in Type 42 fonts. See the corresponding tables 5.1 through 5.3 in *The PostScript Language Reference Manual, second edition*. 
Table 1  Entries in all types of font dictionaries

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontType</td>
<td>integer</td>
<td><em>(Required)</em> Value must be 42.</td>
</tr>
<tr>
<td>FontMatrix</td>
<td>array[6]</td>
<td><em>(Required)</em> Transforms the glyph coordinate system into the user coordinate system. Type 42 fonts, unlike Type 1 fonts, are usually defined in terms of an identity transform, so the value of FontMatrix should be [1 0 0 1 0 0]. See section 2.3 for a discussion of the implications of this choice of coordinate system. FontMatrix must be a literal array.</td>
</tr>
<tr>
<td>FontName</td>
<td>name</td>
<td><em>(Optional)</em> The font program’s PostScript font name, derived from the TrueType name table.</td>
</tr>
</tbody>
</table>

Table 2  Additional entries in all base fonts *(FontType not 0)*

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>array[256]</td>
<td><em>(Required)</em> An array of 256 glyph names ordered by glyph code value. The encoding is most likely to be either the Apple standard encoding or the Windows ANSI encoding, but other encodings will occur. See section 4.4, “Generating the Encoding Vector.” Encoding must be a literal array.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>integer</td>
<td><em>(Optional)</em> See section 4.6, “Generating Unique Identifiers.”</td>
</tr>
<tr>
<td>XUID</td>
<td>array</td>
<td><em>(Optional)</em> Array of integers that uniquely identifies this font or any variant of it. See section 5.8 of The PostScript Language Reference Manual, second edition. XUID must be a literal array.</td>
</tr>
</tbody>
</table>

Table 3  Additional and modified entries in Type 42 fonts

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaintType</td>
<td>integer</td>
<td><em>(Required)</em> 0 for filled glyphs; 2 for stroked glyphs.</td>
</tr>
</tbody>
</table>
StrokeWidth number  
*Optional* The width of the line used to stroke outline fonts (*PaintType* = 2), in glyph coordinates. This number is interpreted in glyph space; see section 2.3, “Implications of The Glyph Coordinate System.”

Metrics dictionary  
*Optional* Width and sidebearing information for writing mode 0. Not normally present in the original definition of a font; adding this dictionary to a font overrides the widths and sidebearings encoded in the glyph definitions in the TrueType font. This dictionary will only affect Type 42 fonts in version number 2013 and greater of the PostScript interpreter. The values in this dictionary are interpreted in glyph space; see section 2.3, “Implications of The Glyph Coordinate System.”

Metrics2 dictionary  
*Optional* Width and sidebearing information for writing mode 1. In general this dictionary is only interpreted by LanguageLevel 1 devices with composite font extensions and all LanguageLevel 2 devices; for Type 42 fonts it is only recognized by PostScript interpreter version 2013 and greater. The values in this dictionary are interpreted in glyph space; see section 2.3, “Implications of The Glyph Coordinate System.”

CDevProc procedure  
Algorithmically derives global changes to a font’s metrics. See *The PostScript Language Reference Manual, second edition*, p. 277. *CDevProc* works the same in a Type 42 font as in a Type 1 font, aside from the different glyph coordinate system; see section 2.3, “Implications of The Glyph Coordinate System.”

CharStrings dictionary  
*Required* Associates glyph names with glyph descriptions. If an entry’s value is an integer, it is used as an index into the TrueType *loca* table, which contains the byte offsets of glyph definitions in the *glyf* table. If the value is a procedure (executable array or packed array), it is interpreted as described in section 5.6.3 of *The PostScript Language Reference Manual, second edition*. This dictionary must have an entry whose key is *.notdef*.

sfnts array  
*Required* An array of one or more PostScript language string objects containing the binary TrueType font. (see section 4.2 for information on the constraints and format).

CIDMap array, string, integer, or dictionary  
*Required* Maps CIDs to glyph indices. If *CIDMap* is a string, it is treated as an array of glyph index values, each of which is *GDBytes* long, stored high-order byte first. If *CIDMap* is an array of strings, the strings are logically concatenated; each must be a multiple of *GDBytes* long. If *CIDMap* is a dictionary, its keys are CIDs and its values are integers representing glyph indices. If *CIDMap* is an integer, it is simply added to the CID to yield a glyph index. See Section 5.4, “Changes to CIDMap.”
GDBytes: integer  (Required if CIDMap is a string or array) Number of bytes representing each glyph index in the CIDMap.

GlyphDirectory: array or dictionary (Optional) See Section 5.5, “GlyphDirectory.”

MetricsCount: integer (Optional) 0, 2, or 4. Default value: 0
See Section 5.7, “MetricsCount.”

2.3 Implications of The Glyph Coordinate System

As indicated in Table 1, a Type 42 font’s glyph coordinate system is typically defined as an identity transform. This is in contrast to a Type 1 font, whose glyph coordinate system is typically defined at a 1000 unit scale relative to user space.

This difference has implications regarding the interpretation of font dictionary entries whose values are defined in glyph space. If a PostScript program adds or changes such entries in a font dictionary, it must choose values that are appropriate for the font’s glyph coordinate system. In particular, values that would be appropriate for a Type 1 font will be 1000 times too large for a Type 42 font.

The font dictionary entries for which this issue arises include:

- The value of StrokeWidth (when PaintType has been set to 2);
- The contents of the Metrics and Metrics2 dictionaries;
- The operands and results of the CDevProc procedure;
- The values of UnderlinePosition and UnderlineThickness in the FontInfo dictionary.
- The values in the FontBBox array.

3 Identifying Interpreters with TrueType Rasterizers

PostScript interpreters with TrueType rasterizers can be identified from the following entry in the device’s PPD file (version 4.1 of the specification):

*TTRasterizer: RasterizerOption
where \textit{RasterizerOption} can be any of the following:

\begin{tabular}{|c|l|}
\hline
None & No TrueType rasterizer is present, and the device is not capable of receiving a downloadable rasterizer. To use a TrueType font on this interpreter, it must be converted to a Type 1 or Type 3 font. \\
Accept68K & No TrueType rasterizer is built-in, but the device has a 680X0-based controller and enough memory to accept a downloadable TrueType rasterizer. (The code to accomplish this is proprietary to Apple Computer, and is not generally available). \\
Type42 & The device has a Type 42 TrueType rasterizer in ROM. \\
\hline
\end{tabular}

A PostScript program can determine whether a LanguageLevel 2 or 3 device supports Type 42 fonts (no LanguageLevel 1 devices support Type 42) by executing:

\begin{verbatim}
42 /FontType resourcestatus (pop pop true) (false) ifelse
\end{verbatim}

which pushes \textit{true} or \textit{false} on the stack depending on whether Type 42 font support is present.

\section{Conversion Issues}

The following sections discuss issues related to converting a TrueType font into a Type 42 font.

\subsection{The FontInfo Dictionary}

The optional \textit{FontInfo} dictionary may be constructed from entries in the name and post tables in the TrueType font. It is not used by the PostScript interpreter, but some PostScript programs may utilize entries such as \textit{UnderlinePosition} and \textit{UnderlineThickness}.

\subsection{The sfnts Array}

In VM, a TrueType font is represented as an array named \textit{sfnts} consists of PostScript string objects which, when concatenated, represent the entire TrueType font. Multiple strings may be required due to the PostScript language implementation limit of 65535 bytes in a string.

When a TrueType font is divided into multiple strings, for compatibility with pre-2013 interpreters, the strings must begin at TrueType table boundaries, or at individual glyph boundaries within the \textit{glyf} table. The TrueType file format requires that tables begin at 4-byte boundaries and that individual glyph
descriptions begin at 2-byte boundaries. Therefore, each string will contain an even number of bytes of TrueType data. There is a practical limit of 64 K on the size of tables (except for the glyf table), until PostScript version 3011.

For compatibility with Type 42 implementations in PostScript interpreter versions prior to 2013, each string must have one additional padding byte appended by adding “00” to the hex data in the file. That is, the length of each string must be odd. The last byte is not logically part of the TrueType font data and is ignored by the interpreter.

The sfnts array is expressed as a series of strings:

```
/sfnts [ <string1> <string2> ... <stringN> ] def
```

In the font file, the strings are made up of lines of hexadecimal characters. The characters in each line may be preceded, followed, and divided by an arbitrary (but consistent) number of white space or control characters (see the additional compatibility constraint in the bullet list below for fonts downloaded to a hard disk).

For Type 42 fonts to be downloaded to a printer’s disk (or other file system hardware), there are specific additional constraints on the text representation of that array in the file. Observing these constraints and beginning the file with the correct comment line (see section 2.1) enables the PostScript interpreter to have dynamic access to the font file on an as-needed basis, which has significant implications for saving VM. If a Type 42 font has the correct initial comment line but does not conform to the constraints listed below, the result will be an invalidfont error on some interpreters.

Although newer versions of the PostScript interpreter are likely to have fewer restrictions on the format of the sfnts array, the following constraints should be used, for backward compatibility purposes, to enable dynamic access to a disk-based font file:

- There may be whitespace and/or control characters between the /sfnts, the ‘[’, and the ‘<’, and between any string’s ‘>’ and the next string’s ‘<’.

- The string should be subdivided into lines of a constant n characters in length and which may be divided by m characters of white space and/or control characters. The numbers n and m must be constant for the entire sfnts array, although it may vary from font to font. Line lengths should also satisfy the Document Structuring Convention (DSC) constraint: $0 < n \leq 255$ (see Appendix G of The PostScript Language Reference Manual, second edition).

- The data encoding technique used in the sfnts array, for example, either ASCII-Hex or binary (see below) must be the same for all strings in a particular font, but may vary among fonts.
The strings in the `sfnts` array may be represented in binary in the same way as may be used for Type 1 charstrings (see section 2.4 in the *Adobe Type 1 Font Format* book). However, fonts using this representation cannot be installed on disk in PostScript interpreter versions prior to 2013. Also, they cannot be safely transmitted across non-binary channels, so fonts in this format should not be embedded in documents. Use of this format should be limited to disk font installer utilities that know something about the capabilities of the PostScript interpreter being accessed.

To represent a string in a binary representation, a PostScript language procedure must be defined with the following code:

```postscript
/RD {string currentfile exch readstring pop} executeonly def
```

Each use of `RD` is followed by exactly one blank character followed by a sequence of binary bytes that are the string contents:

```postscript
n RD ~binary~bytes~ {noaccess def} executeonly def
```

`RD` itself is preceded by an integer `n` which is the number of binary bytes following the `RD` (not including the single blank that follows the `RD`).

The following is an example of a two-element `sfnts` array encoded in this way:

```postscript
/sfnts [
  62135 RD ~62135~binary~bytes~
  12093 RD ~12093~binary~bytes~
] def
```

Each string contains an even number of bytes of TrueType data, followed by one byte of padding which the PostScript interpreter ignores.

### 4.3 Generating The CharStrings Dictionary

The `CharStrings` dictionary for a Type 42 font is a standard dictionary of `key/value` pairs, where the `key` is the glyph’s name (derived from the TrueType font’s cmap table), and the `value` is an index number into the TrueType font’s loca (glyph offsets) table. The value in the key/value pair may also be a PostScript language procedure (executable array or packed array); see section 5.6.3 of the *The PostScript Language Reference Manual, second edition*.

### 4.4 Generating the Encoding Vector

Note The `/Encoding` vector associates a code point with a glyph name. The glyph name is used to look up the glyph’s glyph id in the `Charstrings` dictionary. Glyph names can either be inferred by the encoding used by the font, or can be derived from glyph names in the font’s post table—if the font’s post table is of a version that allows glyph names.
TrueType fonts used in the Windows or Macintosh environments will generally use the encoding specific to that system, such as ANSI for Windows and the Apple encoding for the Macintosh. The platform-specific encoding can be determined by the platform ID number in a subtable of the cmap table. The post table lists glyph names that differ from the platform’s standard encoding. Only versions 1.0, 2.0, and 2.5 of the post table allow the inclusion of glyph names. Formats 3.0 (the one used in most TrueType fonts), and 4.0, do not contain glyph names. If there is no post table in a TrueType font in the Windows environment, the Windows ANSI encoding can be assumed.

When downloading and installing a Type 42 font on a printer’s hard disk, it is essential that the software use a naming convention that is consistent with that used by software on any host system that might be connected to the printer.

4.5 Glyph Mapping and Metrics Access

Figure 1 illustrates how TrueType glyph descriptions and metrics are referenced by character codes and names in a Type 42 font dictionary.

Figure 1 Type 42 Font Glyph Mapping

When a printer driver builds a Type 42 font, it uses the TrueType cmap table to map character codes to glyph indices, which enables it to build the /Charstrings dictionary (which associates PostScript character names with TrueType glyph indices). However, the cmap table is not used by the PostScript interpreter, so it need not be downloaded.

The PostScript interpreter uses the /Encoding array to look up the character name, which is then used to access the /Charstrings entry with that name. The value of that entry is the glyph index, which is then used to retrieve the glyph description’s byte offset in the loca table. The glyph index is also used
to get metrics from the hmtx. As of PostScript interpreter version 3011, the
glyph index can also be used to get vertical metrics from the vmtx table (see
section 5.6, “Vertical Writing Mode”).

4.6 Generating Unique Identifiers

The Type 42 font may contain an unique identifier which allows the glyph
bitmaps to be cached across print jobs (see also section 5.8 of The PostScript
Language Reference Manual, second edition). This entry is optional but
highly desirable since many users may use the same fonts in every print job.

Bitmaps generated from TrueType fonts in Type 42 format use the same
caching system as is used for Type 1 fonts. When a glyph bitmap is needed
from a Type 42 font, the glyph cache is checked first. If the bitmap has not
been cached, the bitmap is produced from the outline font program.

TrueType fonts do not contain any type of unique number which either
corresponds to the PostScript language UniqueID entry or could be used for
such. Using something like a checksum number as a UniqueID value devices
would not be advisable since it does not assure uniqueness. Although this
approach would work in many situations, there is an increased and
unacceptable risk when, as at a service bureau, bitmaps are cached on a hard
disk for a potentially long period of time. Hence, the performance gain
resulting from caching does not offset the danger of a user getting incorrect
bitmaps from the cache.

Since TrueType rasterizers only exist in LanguageLevel 2 interpreters, the
XUID operator offers a safer opportunity to cache bitmaps. The XUID
(extended unique ID) is an array of integers which provides for distributed,
hierarchical management of UniqueID numbers. The goal is to have a
mechanism for generating an XUID array of values, on-the-fly, which are
unique for every font, yet exactly repeatable since a TrueType font in a user’s
system may be converted multiple times to a Type 42 for printing.

A recommended method for generating a number for a given font which is
both more likely to be unique than a simple checksum and exactly repeatable,
is to use the MD5 algorithm from RSA Data Security, Incorporated. Their
software can be copied and freely distributed if it is properly identified. The
code for this algorithm is readily available from:

RSA Data Security, Inc.
100 Marine Parkway
Redwood City, CA 94065

The goal is to generate an XUID array of five elements, with the first having
the value of 42 (decimal). This value has been registered in the Adobe XUID
registry for use by software in creating Type 42 fonts. The MD5 algorithm
can then be used to generate a 128-bit number, using the font file as input. This number can then be divided into four 32-bit integers to make the other four elements of the array. Some optimization of the algorithm code may be necessary to enhance performance.

4.7 Required TrueType Tables

In creating a Type 42 font from a TrueType font, only a subset of all potential tables in the original font are actually used by the rasterizer in the PostScript interpreter. The following tables are the set of tables that are used by the TrueType rasterizer. Not all of these tables are required for every configuration of a Type 42 font.

<table>
<thead>
<tr>
<th>head</th>
<th>prep</th>
<th>cvt_</th>
</tr>
</thead>
<tbody>
<tr>
<td>hhea</td>
<td>fpgm</td>
<td>maxp</td>
</tr>
<tr>
<td>hmtx</td>
<td>vhea</td>
<td>vmtx</td>
</tr>
<tr>
<td>loca</td>
<td>glyf</td>
<td></td>
</tr>
</tbody>
</table>

The tables vhea and vmtx are *only* supported in version 3011 or higher, and only when MetricsCount has a value of 0 or 2, or if the MetricsCount keyword is not included. Also, for compatibility reasons, CDevProc should be included for vertical writing mode.

Since a significant number of tables may be included in a TrueType font (including potentially large kerning and metrics tables), performance may be improved by including in the downloadable Type 42 font only the tables actually used by the TrueType rasterizer.

5 CIDFontType 2 CID Fonts

Version 2015 of the PostScript interpreter introduced support for large Chinese, Japanese, and Korean (CJK) multi-byte TrueType fonts by supporting the use of CID fonts that contain TrueType glyphs. The CID font format allows thousands of glyphs to be encoded and accessed. The CID font must have the keyword/values CIDFontType 2, and FontType 42. For information on the CID format, see Technical Note #5014, Adobe CMap and CIDFont Files Specification.

For downloading and installing a CIDFontType 2 CID font to a printer’s hard disk, the font should contain the TrueType font embedded in the sfnts array. The number of tables included can be limited to only those tables used by the rasterizer (see section 4.7, “Required TrueType Tables”). The inclusion of other tables depends on the desired optimizations for printing speed, or whether the font is to be permanently downloaded to a printer’s hard disk. For example, for vertical writing, a GlyphDirectory may be included in the
PostScript portion of the font if the font is to be subsetted or incrementally downloaded; but for disk-based fonts, including the vhea and vmtx tables in the sfnts array gives better performance.

A large CJK TrueType font may have 35,000 glyphs, as well as a number of very large data tables. For a CIDFontType 2 CID font that is to be downloaded for printing a specific document, a number of optimizations can be made to limit the amount of data that must be downloaded. For example, the font can be subsetted to include only the glyphs needed in the document, and those glyphs can be downloaded incrementally on an as-needed basis. These optimizations can improve performance as well as save VM, which helps to ensure that a document will print when memory is low.

5.1 Complete Font Downloading

When downloading a TrueType font to a printer’s hard disk, the optimizations for subsetting and incremental downloading are not used because the full font must be available.

Figure 2 shows the configuration for glyph mapping for TrueType fonts to be permanently downloaded to a printer’s hard disk. It is desirable to have a full CIDMap, as well as full hmtx, vmtx, loca, and glyf tables. The TrueType cmap table is not shown because it is not needed by the rasterizer, but it is used initially by the driver software to get the glyph indices for building the /Charstrings dictionary.

The interpreter maps a character code to a CID number, and uses the CID to get the TrueType glyph index from CIDMap. The glyph index is then used to get horizontal or vertical metrics from the hmtx or vmtx tables, respectively. A glyph description is obtained from the glyf table by getting its byte offset from the loca table.
5.2 Incremental Font Downloading

Figure 3 shows an example of how glyphs are accessed for CIDFontType 2 CID fonts that use a GlyphDirectory for subsetting and incremental downloading. The CMap resource is used to map the character code to a CID number, and the CIDMap maps CIDs to glyph indices. If a special CMap can be constructed which makes CIDMap an identity mapping, then downloading the identity CIDMap is wasteful and unnecessary. In this case, CIDMap can simply be defined as an integer. The glyph index is then used to access the glyph description in GlyphDirectory.
5.3 CMap Resources for CIDFontType 2 CID Fonts

CIDFontType 2 fonts must have an associated CMap resource. Since the original TrueType font does not have a CMap file, the driver or downloader software must create one.

The CMap resource (maps character codes to CIDs) can be constructed so that the mapping in CMap (CIDs to glyph indices) is an identity mapping, such that:

\[ GI_i = CID_i \]

Because the resulting CMap uses an identity mapping, it is not necessary to download CMap as a dictionary or array, thus saving a significant amount of VM. Although CMap is a required entry, it can be defined as having an integer value (see “Defining CMap as an Integer” in section 5.4). In the general case, the value of the integer would be zero.

A CID-keyed font must reference the /Registry, /Ordering, and /Supplement specified in the CMap resource. Hence, both the CIDFont and the CMap must be built to reflect that relationship (see Adobe Technical Note #5014, Adobe CMap and CIDFont Files Specification).

5.4 Changes to CMap

Prior to interpreter version 3011, a CMap could be either a string or an array of strings, and was interpreted as a mapping of CIDs to glyph indices. The resulting glyph index is then used to access glyphs in either the glyf table or the GlyphDirectory, as well as metrics data in the hmtx or vmtx tables in the TrueType font.
For large multi-byte fonts, CIDMaps tend to be fairly large, and if only a fraction of the font’s glyphs are needed for a particular document, storage space can be wasted. For example, the Windows Simplified Chinese font SimHei has about 25 K glyphs and its CIDMap requires about 50 KB of file size and VM. The following sections explain how to define CIDMap as a dictionary or an integer, so only a subset of all glyph mappings need to be downloaded. Alternatively, it can be defined as an integer, so that the CIDMap need not be downloaded at all (except as a single value).

### Defining CIDMap as a Dictionary

Defining CIDMap as a dictionary facilitates incremental downloading of glyphs by allowing the downloading of only the mapping entries needed by the current document. This can save a significant amount of both file and VM space.

### Defining CIDMap as an Integer

CIDMap can be defined as having a single integer value:

```
/CIDMap integer
```

where integer is interpreted as an offset that is added to the CID obtained from the CMap resource, to map to the corresponding glyph index value:

```
GI = integer + CID
```

For the general case, the value of the integer will be zero. It may also sometimes be necessary to create a CIDMap in which there is an offset from the Identity mapping. For example, if it is attempted to download a TrueType font with 20,000 glyphs to a printer’s hard disk, the large loca table (20K x 4 bytes/entry) will not be interpreted correctly if it is split into multiple sfnts strings. The font can be downloaded as two CIDFonts, one with the glyph’s from 0 to n–1; and the other with glyphs from n to 20K–1 (being careful not to split composite glyphs across two fonts). Hence, a glyph index used in a host document can be looked up using the CMap (where GI = CID) to decide which CIDFont to use when printing.

### 5.5 GlyphDirectory

Both Type 42 and CIDFontType 2 CID fonts may be created with a GlyphDirectory array or dictionary, which allows font subsetting and incremental downloading.

For a CIDFontType 2 font, if GlyphDirectory is a dictionary, each key is an integer glyph index, and the value is a string containing the TrueType glyph description. If GlyphDirectory is an array, its length must be greater than the
highest glyph index in the font. Each array element can be either null (indicating an empty element), or a string containing the TrueType glyph description.

Starting with PostScript version 2015, the interpreter checks for the existence of a \texttt{gdir} table in the \texttt{sfnts} array, and if found, uses \texttt{GlyphDirectory} in place of the \texttt{loca} and \texttt{glyf} tables. This table, which is not in the TrueType font format specification, must be inserted by the software that builds the \texttt{CIDFontType 2} font. The size and offset of the \texttt{gdir} table must be zero, it is only used to specify the use of the \texttt{GlyphDirectory} entry in the font dictionary.

The \texttt{loca} or \texttt{hmtx} tables are of the order of magnitude of $4 \text{ bytes} \times \text{numGlyphs}$. Thus, for incrementally downloaded or subset fonts, the choice to use \texttt{GlyphDirectory} avoids the need to download a large and mostly unused \texttt{loca} table, and saves the corresponding space in VM. For PostScript versions before 3011, for these same small fonts, the large and mostly unused \texttt{hmtx} table must still be downloaded. In 3011, the horizontal and vertical metrics may be included in the \texttt{GlyphDirectory}, making it unnecessary to download the additional tables (see section 5.7, “MetricsCount”).

For large \texttt{CIDFontType 2} CID fonts that are to be permanently downloaded in full to a PostScript file system, \texttt{GlyphDirectory} should not be used because it is less efficient than using the \texttt{loca} and \texttt{glyf} table method. In addition, for compatibility with earlier PostScript versions, some font downloading applications may choose to download the \texttt{hmtx} table (and, now, \texttt{vmtx} table) rather than a \texttt{GlyphDirectory}, in order to provide compatibility with interpreters before 3011.

Once a \texttt{GlyphDirectory} array or dictionary is defined, a PostScript program may insert entries containing new glyph descriptions by replacing null array elements with strings or inserting new dictionary entries. It may not replace entries that are not null; due to font caching, any attempts to do so will yield unpredictable results. If a glyph is added to \texttt{GlyphDirectory} between \texttt{save} and \texttt{restore} operations, the \texttt{restore} operator will remove it, since this is equivalent to replacing an entry that is not null. To avoid this problem, the driver must download the same glyph definition again before the next attempt to perform a \texttt{show} operation on that glyph.

As is true for all CID fonts, an attempt to perform a \texttt{show} operation on a glyph whose CID selects a null or missing entry in the \texttt{CMap} resource, the \texttt{CMap} will be consulted to see if there is a .notdef CID for that character code. If there is, it will be used; otherwise a CID of 0 will be used. The \texttt{GlyphDirectory} entry for CID 0 must be present and not null, or an \texttt{invalidfont} error will occur.

If \texttt{GlyphDirectory} is an array, it must be allocated with enough entries to store the highest CID or glyph index that is expected. Any unused entries in the array will be wasted space. An array of a given length consumes about 40
percent of the memory used by a dictionary of the same length. Thus, the
dictionary representation is advisable only for a sparsely populated font
containing less than 40 percent of the total glyphs in the font.

5.6 Vertical Writing Mode

CJK fonts are typically used in either horizontal or vertical typographic
modes. For vertical layout, the metrics data is available in the host font’s vmtx
table (advance height and top side bearing). It is up to the print driver to
decide whether both horizontal and vertical metrics need to be included in the
CIDFontType 2 CID font that is to be downloaded.

/WMode 1 indicates that the renderer should use vertical metrics. Thus, if
MetricsCount is 0 or 2 (see section 5.7, “MetricsCount” below) the metrics
data is found in the vmtx table, which must be provided. If MetricsCount is
missing (the default value is 0, which indicates that no metrics are contained
in GlyphDirectory), it is up to the creator to provide the metrics using
CDevProc or Metrics2, or both. If MetricsCount is 4, the data is found in the
first through fourth bytes at the start of the glyph description.

5.7 MetricsCount

The addition of the MetricsCount key in 3011 allows the inclusion of metrics
data in the GlyphDirectory dictionary or array. This has the advantage that
only the metrics for the subsetted characters need be included, rather than
having to download complete hmtx and vmtx tables.

When a key /MetricsCount is found in a CIDFont with CIDFontType 2, it
must be an integer with values 0, 2, or 4.

Prior to 3011, the TrueType rasterizer in a PostScript interpreter ignored
metrics specified for vertical writing mode in the vmtx table, and vertical
metrics had to be supplied using a Metrics2 dictionary. For a full font
downloaded to PostScript file system, the full Metrics2 dictionary is large and
takes up a large amount of VM. Even for a font which is incrementally
downloaded or subsetted, the Metrics2 overhead is inefficient. With 3011, the
vertical metrics can be supplied using a vmtx table, or they can be embedded
in the GlyphDirectory entries for incremental downloading.
Table 4 shows where the metrics data is located, based on the value of MetricsCount.

### Table 4  *Metrics Location based on MetricsCount Value*

<table>
<thead>
<tr>
<th>MetricsCount</th>
<th>Number of metrics data bytes</th>
<th>Horizontal Metrics</th>
<th>Vertical Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>hmtx or <strong>Metrics</strong> dictionary</td>
<td>vmtx, <strong>Metrics2</strong> dictionary, or <strong>CDevProc</strong></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td><strong>GlyphDirectory</strong> (4 bytes preceding the glyph description)</td>
<td>vmtx, <strong>Metrics2</strong>, or <strong>CDevProc</strong></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td><strong>GlyphDirectory</strong> (second set of four bytes preceding the glyph description)</td>
<td><strong>GlyphDirectory</strong> (first set of four bytes preceding the glyph description—see Table 5)</td>
</tr>
</tbody>
</table>

**MetricsCount** may have a value of 0, 2, or 4. A value of 0 indicates that no metrics data is included in the **GlyphDirectory**, and hmtx and/or vmtx tables are included in the font. If **MetricsCount** is missing, a value of 0 is implied.

If **MetricsCount** is 2 or 4, it specifies that horizontal, or vertical and horizontal metrics, respectively, precede the glyph description in the **GlyphDirectory** dictionary or array. These metrics values are copied directly from the TrueType glyph description, without any conversion or translation.

In all cases, the glyph description will begin \((2 \times \text{MetricsCount})\) bytes from the beginning of the string. If **MetricsCount** has a value of 2, the four bytes of data will be the horizontal advance width and left sidebearing. If **MetricsCount** has a value of 4, then there will be eight bytes of metrics data: the vertical advance height and top sidebearing, followed by the horizontal advance width and sidebearing, with the first byte of each pair being the high order byte.

### Table 5  *Metrics Data in Glyph Directory*

<table>
<thead>
<tr>
<th>MetricsCount</th>
<th>byte #</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0,1</td>
<td>horizontal advance width</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>left sidebearing</td>
</tr>
</tbody>
</table>
6 Known Bugs

There is a known bug in the TrueType rasterizer included in versions of the PostScript interpreter previous to version 2013. The problem is that the translation components of the FontMatrix, as used as an argument to the definefont or makefont operators, are ignored. Translation of user space is not affected by this bug.

7 Example Type 42 single-byte font program

```
%!PS-TrueTypeFont-65536-65536-1
11 dict begin
/FontName /Chicago def
/Encoding 256 array
0 1 255{1 index exch/.notdef put}for
dup 0 /.null put
dup 1 /option put
dup 2 /control put
%
%... many Encoding array entries omitted...
%
dup 253 /hungarumlaut put
dup 254 /ogonek put
dup 255 /caron put
readonly def
/PaintType 0 def
/FontMatrix [1 0 0 1 0 0 ] def
/FontBBox[-190 -283 1164 1090 ] def
/FontType 42 def
/XUID [42 16#7880BE99 16#AC61C9D 16#D021DE98 16#1F9CD56] def
%
% Optional FontInfo dictionary may be inserted here
%
/fsfts[<
000100000000900090000009
637674202B194DE00000009C00000290
6670676D31773E0000000032C0000003B6
%
%...many sfnts lines omitted...
%
5BB0F1D594569534273373737373737345684400
0D>]def
/CharStrings 279 dict dup begin
/.notdef 0 def/.null 1 def/nonmarkingreturn 2 def
/space 3 def/exclam 4 def /quotedbl 5 def/numbersign 6 def
/dollar 7 def/percent 8 def/ampersand 9 def
%
```

Table 5  Metrics Data in Glyph Directory

<table>
<thead>
<tr>
<th>Metrics Count</th>
<th>byte #</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0,1</td>
<td>vertical advance width</td>
</tr>
<tr>
<td>2, 3</td>
<td></td>
<td>top sidebearing</td>
</tr>
<tr>
<td>4, 5</td>
<td></td>
<td>horizontal advance width</td>
</tr>
<tr>
<td>6, 7</td>
<td></td>
<td>left sidebearing</td>
</tr>
</tbody>
</table>
%...many CharStrings entries omitted...
%
/checkmark 273 def/linebreakltor 274 def
/linebreakrtol 275 def /markingnobreakspace 276 def
/diamond 277 def/appleoutline 278 def end readonly def
FontName currentdict end definefont pop
# Index

## Symbols
- `notdef` 5

## B
- bitmap cache 11
- bugs 20

## C
- `CDevProc` 5
- character coordinate system 6
- comment lines 3
- conversion issues 7

## F
- `FontInfo` dictionary 4, 6
- `FontMatrix` 20
- `FontType` 4, 7

## G
- glyph coordinate system 4, 6

## H
- hard disk 2, 3

## I
- `invalidfont` error message 8

## M
- `makefont` 20
- MD5 algorithm 11
- `Metrics` 5
- `Metrics2` 6

## P
- `PaintType` 4
- PPD file 2, 6

## R
- `resourcestatus` 3

## S
- `sfnts` 2, 8
- string
  - binary representation 9
- `StrokeWidth` 6

## T
- TrueType rasterizer 2, 6, 7, 20
- TrueType specification 1
- TrueType table
  - `cmap` 9, 10
  - `cvt` 12
  - `glyf` 5, 12
  - `head` 12
  - `hhea` 12
  - `loca` 5, 9
  - `name` 4, 7
  - `post` 3, 7, 10
  - `prep` 12
- Type 42 Font Dictionary 3

## U
- `UnderlinePosition` 7
- `UnderlineThickness` 6, 7

## V
- `VM` 2, 3
VMusage comment 3