Adobe PostScript™

CID-Keyed sfnt Font File Format for the Macintosh

Adobe Developer Support

Version 2.0

Technical Note #5180

12 February 1997

Adobe Systems Incorporated

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

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

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

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

PN LPS5180

Contents

CID-Keyed sfnt Font File Format for the Macintosh

1 Introduction

This document describes the CID sfnt font format for the Macintosh. It describes only those extensions made by Adobe to Apple's format to accommodate CID-keyed fonts. Also, it assumes familiarity with the sfnt specification published by Apple Computer (see Appendix B).

A CID-keyed sfnt font consists of a PostScript CID-keyed font program (the CIDFont) embedded in a subtable of a sfnt wrapper. The CID-keyed format was designed for maximum flexibility and performance for large character set fonts such as for Chinese, Japanese, and Korean (CJK) language fonts. The format supports both regular and rearranged CID-keyed fonts.

For more information on the CID-keyed font file format, see Adobe Technical Note #5014, "Adobe CMap and CIDFont Files Specification," and #5092, "CID-Keyed Font Technology Overview." Also, a CID Software Developers Kit (SDK) is available from the Adobe Developers Association.

This document describes only a few tables beyond the minimum number required for a CID sfnt font. For example, the ALMX table allows the specification of alternate metrics for proportionally spaced characters. Also, a number of other advanced features can be supported by using the mort table (the format of the mort table is specified in the Apple documentation).

This document also includes:

- Appendix A: sfnt Data Types
- Appendix B: Bibliography
- Appendix C: Glossary
- Appendix D: sfnt Binary Search and Lookup Table Formats

1.1 Purpose of this Document

This document describes how to make CID-keyed fonts for use in future Macintosh operating systems which are likely to support only fonts in the sfnt format. Font developers who currently have CID-keyed fonts, or PostScript language OCF (Original Composite Format) fonts, must convert these into the sfnt format for use with future Macintosh systems. OCF fonts can be converted to the CID-keyed font file format, and a CID-keyed font can be converted into the sfnt format by embedding the CIDFont into a subtable of the sfnt resource.

For the future, the best way to develop full-featured fonts for the broadest possible market will be to use the new OpenType™ font format, which is being jointly developed by Adobe Systems and Microsoft Corporation. For the immediate future, developers who wish to make multi-byte fonts for the upcoming Macintosh operating systems should use the CID sfnt format.

Both the sfnt and OpenType formats are essentially font packaging formats which allow the embedding of a CID-keyed font as a single subtable. This means that it should be relatively easy, in the future, for font developers to convert CID sfnt fonts into the OpenType format.

1.2 Conventions Used in this Document

In this document, all sfnt table names and table elements are represented in a sans serif typeface, and PostScript language and Type 1 font program operators are represented in a bold sans serif style.

References to documents include full information in the first reference, and subsequent references may use a shortened form. Please refer to Appendix B for complete bibliographic information.

2 The CID sfnt Format

A CID sfnt font consists of an sfnt "wrapper" that contains a complete CIDFont as a subtable. The tables in the sfnt wrapper are identified by unique four-character tags – for example, 'cmap'. The names of all data tables defined by Apple use all lowercase characters. All non-Apple tables developed by Adobe are in all uppercase letters – for example,'CID'.

A CID sfnt font is distinguished from a TrueType font by having the tag 'typ1' stored in the 4-byte Fixed version field of the sfnt header (see Table 1), whereas a TrueType font has either the tag 'true', or the value 1.0. A CID-keyed Type 1 sfnt font is distinguished from a Type 1 Roman sfnt font by the presence of a CID table, and the absence of a TYP1 table.

Conceptually, there are three categories of sfnt tables. The system-specific tables, such as name and cmap, are required by every font in order to register it and enable the operating system to use it. The optional layout-specific tables are those used to format text (which are not described in this document). Finally, the rasterizer-specific tables are those used by either the Type 1 rasterizer, or the TrueType rasterizer, to rasterize glyphs, generate glyph outlines, prepare the font for downloading, or return kerning information. The tables described in this document are those in the first and third categories.

Note *A CID sfnt font does not use the form of the CMap file described in Adobe Technical Note #5014, "Adobe CMap and CIDFont Files Specification." Instead, the Apple-defined* cmap *table is used for character encoding.*

2.1 Compatibility

A CID sfnt font defined by this specification will work with both QuickDraw and QuickDraw GX. The font will work with QuickDraw by having an FNAM table, which also allows Adobe Type Manager® software (ATM®) to use QuickDraw NFNT resources to display bitmaps for QuickDraw GX. The FNAM table is also used to select a cmap subtable for encoding. For more information on how encodings are selected in compatibility mode, see section 12.1.

To print CID-keyed sfnt fonts on a PostScript language printer, the CID-keyed font must be extracted from the sfnt wrapper. This can be accomplished by getting the offset and length of the CIDFont in the sfnt resource Table Directory (see Figure 2).

The CID sfnt format also allows the use of rearranged CID-keyed fonts; for more information, see section 12.2.

2.2 Glyph Transformations

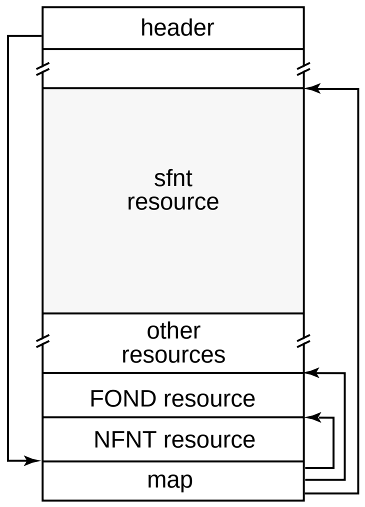
The CID sfnt format allows the specification of a set of transformations that can be applied to the glyphs in the font. For CJK fonts, this may include features such as ligature substitution and the specification of alternate metrics for proportionally spaced glyphs.

These glyph transformations are achieved by using the glyph metamorphosis table (tag name: mort), which is defined and documented by Apple. In the case of providing alternate metrics for proportionally spaced glyphs, the CID-specific table ALMX (Alternate Metrics) table, defined by Adobe and described in section 4 of this document, must be used.

3 CID sfnt Font Format Overview

Macintosh sfnt fonts are required to be part of a suitcase (an FFIL resource file). Additionally, the suitcase must contain at least one FOND resource, and may optionally contain other resources such as bitmap fonts (NFNT resources) for screen display.

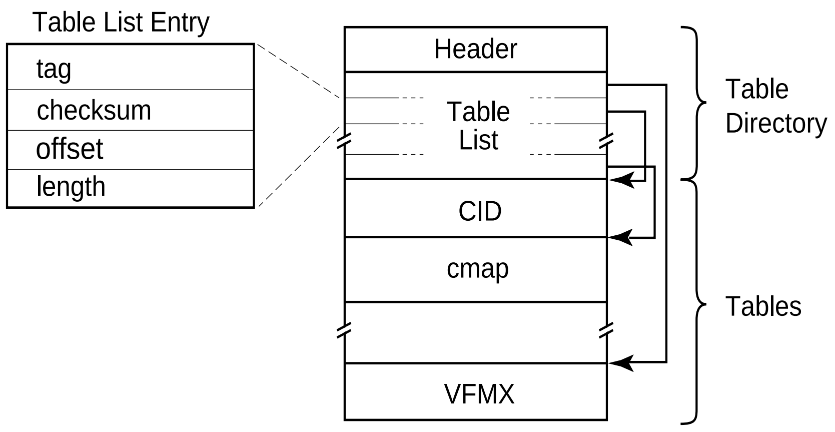
Figure 1 *sfnt Font File Organization*



Information on how Macintosh resources are constructed is contained in *Inside Macintosh – More Macintosh Toolbox;* Chapter 1: Resource Manager.

Figure 2 shows the contents of the sfnt resource. The tables may be in any order, but the Table List entries must be sorted into tag order.

Figure 2 *Layout of the sfnt*



The contents of the sfnt Header are shown in Table 1.

Table 1 *sfnt Header*

Type	*Field*
Fixed	version
uint16	numTables
uint16	searchRange
uint16	entrySelector
uint16	rangeShift

The version field originally contained the sfnt format version number, 1.0 (0x00010000). However, the purpose changed with the introduction of QuickDraw GX, and this field is now used to identify a rasterizer for the font. Type 1 fonts use the tag 'typ1'; older TrueType fonts use the value 1.0 (0x00010000), and newer TrueType fonts use the tag 'true' as the value for this field.

The numTables field specifies the number of entries in the Table List (where each entry has four fields, as shown in Table 2).

The searchRange, entrySelector, and rangeShift fields provide values intended to speed the binary search of the Table List. Please consult "QuickDraw GX Font Formats: The TrueType Font Format Specification" for more information.

Following the sfnt Header is the Table List, where each entry identifies and points to one table in the sfnt as shown in Table 2.

Table 2 *Table List Entry*

Type	*Field*
uint32	tag
uint32	checksum
uint32	offset
uint32	length

The tag field identifies the table by using an ASCII representation of the table's name. Table names shorter than four characters are padded with the code for a blank character.

The checksum field contains a checksum of all bytes contained in the table.

The offset field contains the byte offset from the beginning of the sfnt to the start of the table.

The length field contains the length of the table in bytes. Note that each sfnt table is padded to a long-word (4-byte) boundary.

3.1 Guidelines for Version Checking

Version numbers for individual sfnt tables are stored as 4-byte Fixed values and are of the form *x.y*, where *x* is the major version number, and *y* is the minor version number. Following Apple's convention, the version numbers are represented in a hexadecimal format; for example, version 1.1 is represented as 0x00010001.

Applications should always check the version number of all sfnt tables before they are used. Each revision or extension of a table will be documented in this specification.

If the major version number is greater than the version understood by the application, the table should not be read and the application should take the appropriate action. Minor revisions are those that allow the application to safely ignore, for example, fields added to the end of a table. If differences in the minor version are found, some information may not be used, but use of the font file can continue safely.

3.2 Overview of CID sfnt Tables

A CID sfnt font must contain the cmap, fdsc, and name tables. The following tables are specific to CID-keyed sfnt fonts:

ALMX (required if proportionally-spaced characters are included in the font)
The Alternate Metrics table (ALMX) specifies the metrics for proportionally spaced characters.

BBOX (required)
This table specifies bounding boxes for all glyphs in the font.

CID (required)
This table contains the CIDFont file. For a rearranged CID font, it contains encoding information rather than glyph descriptions.

COFN (required for rearranged CID sfnt fonts)
Table used only for rearranged CID sfnt fonts; it contains the QuickDraw font name for each component font used in the rearranged font.

COSP (required for rearranged CID sfnt fonts)
This table, used only for rearranged CID sfnt fonts, defines the code space of a rearranged font.

FNAM (required for QuickDraw compatibility)
This table maps FOND resource names and styles and is used for compatibility with QuickDraw.

HFMX (required)
This table provides font metric information related to setting horizontal lines of text. These values are font-wide metrics such as ascent, descent, line gap and caret information values.

VFMX (required)
This table provides vertical metric information such as the per-glyph vertical advance and origin shift information.

The following sections give a more detailed description of the CID-keyed font–specific tables in a CID sfnt font. The tables are discussed in alphabetic order.

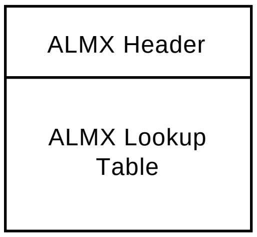
Note *A number of tables contain fields that duplicate information stored elsewhere in the font. While redundant, this makes these tables more self-contained and simplify their use.*

4 ALMX

The ALMX table provides for alternate metrics that override the existing metrics of the glyphs in the CID-keyed font. The new glyphs defined in this table are intended to be used with non-contextual glyph substitution for proportional Japanese fonts.

The ALMX table consists of a header followed by a Lookup table, as shown in Figure 3. The Lookup table may be in format 0, 2, 4, 6, or 8, as described in Appendix E.

Figure 3 *The ALMX Table*



The format of the ALMX Header table is shown in Table 3.

Table 3 *ALMX Header Table*

Type	*Field*
Fixed	version
uint16	flags
uint16	nMasters
uint16	firstGlyph
uint16	lastGlyph

The version field should be set to 1.0 (0x00010000).

The flags field is reserved for future use, and should be set to zero.

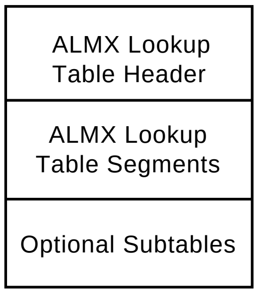
The nMasters field must have a value of 1 (for non-multiple master fonts).

firstGlyph is the first index of valid glyphs specified in the lookup table.

lastGlyph is the last index of valid glyphs specified in the lookup table.

The ALMX Lookup Table consists of a header followed by a lookup data section, as shown in Figure 4.

Figure 4 *ALMX Lookup Table*



The format of the ALMX Lookup Table can be any one of the five lookup table formats; Table 4 shows the form used for formats 0, 2, 6, and 8.

Table 4 *ALMX Lookup Data for Formats 0, 2, 6, and 8*

Type	*Field*
ALMXDataEntry	ALMXDataEntry

The contents of the ALMXDataEntry is shown in Table 5. This format is used for the Lookup Table segments as well as for any optional subtables that may be present.

Table 5 ALMXDataEntry

Type	*Field*
int16	glyphIndexOffset
FUnit	horizontalAdvance
FUnit	xOffsetToHOrigin
FUnit	verticalAdvance
FUnit	yOffsetToVOrigin

glyphIndexOffset is the offset to the glyph index of the glyph whose metrics are being modified.

horizontalAdvance specifies the horizontal advance of the glyph in the horizontal writing mode (specified in FUnits).

xOffsetToHOrigin specifies the horizontal offset from the original origin to the new origin in the horizontal writing mode (specified in FUnits).

verticalAdvance specifies the vertical advance of the glyph in the vertical writing mode (specified in FUnits).

yOffsetToVOrigin specifies the vertical offset from the original origin to the new origin in the vertical writing mode (specified in FUnits).

Table 6 *ALMX Lookup Data for Format 4*

Type	*Field*
uint8	flags
uint24	offsetToSubTable

The flags field is defined as shown in Table 7.

Table 7 *flags Values for the ALMX Lookup Data*

Name	*Value*
ONE_ENTRY_IN_SUBTABLE	0x80

The flags field may have a flag ONE_ENTRY_IN_SUBTABLE set to indicate that there is only one entry in the subtable. The other bits are reserved for future use, and should be set to zero.

The offsetToSubTable field contains the offset to the subtable, which is shown in Table 8.

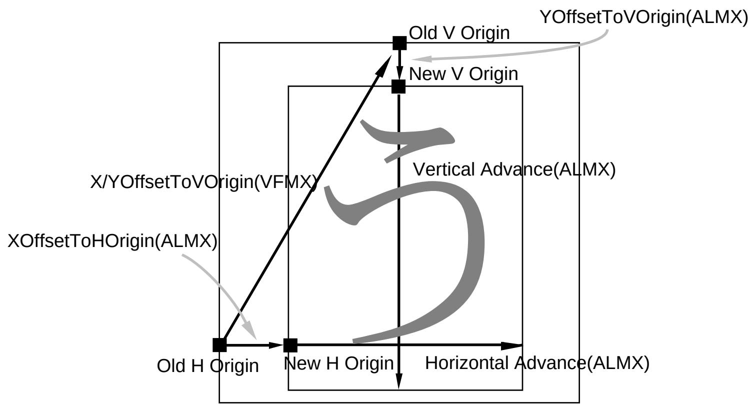
Table 8 *ALMX SubTable*

Type	*Field*
ALMXDataEntry	ALMXSubTableEntry[<variable>]

The number of entries contained in ALMXSubTableEntry for a lookup segment depends on the value of the flags bit. Each lookup data has an offset from the start of the lookup table to the subtable which contains the metrics information. When the ONE_ENTRY_IN_SUBTABLE flag is set in flags, there is only one entry in the subtable, and the metrics in the entry applies to all glyphs in the segment. When this bit is not set, the subtable has one entry for each glyph in the segment, and it is indexed by the glyph index minus the first glyph index in the lookup segment.

Figure 5 shows the character metrics values associated with a glyph referenced by an entry in the ALMX table.

Figure 5 *Alternate Metrics Example*



Typical uses of the ALMX table are as follows. When GlyphIndexOffset is zero, the CID glyph with the same glyph index used to index this table is redefined to have the new metrics specified in this table. When the GlyphIndexOffset is not zero, a new glyph is defined based on the glyph whose glyph index is calculated by adding GlyphIndexOffset to the glyph index of this glyph.

If the listed metrics value is greater than 32000, the default metrics of the specified glyph charstring are used.

If a glyph has metrics specified in both the ALMX and VFMX table, the metrics specified in the ALMX table override those in the VFMX table.

5 BBOX

The BBOX table specifies the bounding boxes for all the glyphs in the font.

Table 9 *BBOX Table*

Type	*Field*
Fixed	version
uint16	flags
uint16	nGlyphs
uint16	nMasters
BBox	bbox[nGlyphs]

The version field should be set to 1.0 (0x00010000).

The flags field is reserved for future use, and should be set to 0.

The nGlyphs field specifies the number of glyphs in the font. Its value is equal to the CIDCount field for a CID-keyed font.

The nMasters field must have a value of 1 for non–multiple master fonts.

bbox is an array of BBox elements, described below.

Table 10 *BBox Element*

Type	*Field*
FWord	left
FWord	bottom
FWord	right
FWord	top

The metric left is the minimum x-coordinate of the glyph bounding box.

The metric bottom is the minimum y-coordinate of the glyph bounding box.

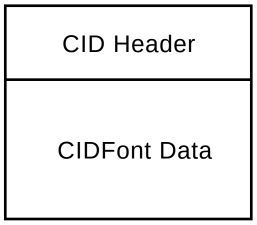
The metric right is the maximum x-coordinate of the glyph bounding box.

The metric top is the maximum y-coordinate of the glyph bounding box.

6 CID

The CID table consists of two parts, the CID Header table, and the CIDFont data, which contains the glyphs. The layout of the CID table is shown in Figure 6.

Figure 6 *CID Table Layout*



The contents of the table depends on whether the CID font is a regular CID-keyed font, or a rearranged CID-keyed font. The format for a regular CID-keyed sfnt font is shown in Table 11.

Table 11 *CID Header Table*

Type	*Field*
Fixed	version
uint16	flags
uint16	CIDCount
uint32	CIDFontLength
uint32	asciiLength
uint32	binaryLength
uint16	FDCount

The version field should be set to 1.0 (0x00010000).

The flags field bits are defined in Table 12.

Table 12 *flags Values for the CID Table*

Name	*Value*	*Description*
CID_REARRANGED_FONT	0x0001	Specifies a rearranged CID font

The flags field may have a flag CID_REARRANGED_FONT set to indicate that the font is a rearranged font. If not set, the font is a CID font. The other bits are reserved and should be set to zero.

Continuing with the entries in the CID Header table, CIDCount must be equal to the value specified by the **CIDCount** key in the CIDFont. For a rearranged CID sfnt font, this value must be set to the total number of glyph indices assigned in the rearranged font.

CIDFontLength is the total length of this table excluding the header part.

AsciiLength is the length of the ASCII part of the CIDFont, and it is equal to the file offset to the beginning of the binary section of the CIDFont.

BinaryLength is the length of the binary section of the CIDFont.

FDCount is the number of font dictionaries defined in the FDArray of the CID-keyed font.

6.1 CID Table for a Rearranged CID Font

If the CID_REARRANGED_FONT flag is set, then the font is a rearranged CID font. For a rearranged CID font, the AsciiLength specifies the length of the rearrangement recipe part that begins with *%ADOStartRearrangedFont* and terminates with *%%EndResource*. An example of a rearranged CID sfnt font is shown in Example 1, shown below.

BinaryLength field specifies the length of the rearranged font procset portion at the beginning of the font.

Note *A rearranged font, unlike a CIDFont, has a binary section (the ProcSet portion) before the ASCII section. It does not contain binary charstrings, it only references those in the component CIDFont files.*

The other fields, CIDCount and FDCount, are not used and should be set to zero.

The following is an example of a rearranged CID font (for more information, see Adobe Technical Note #5014, "Adobe CMap and CIDFont Files Specification 1.0.").

In this example, the template font is HeiseiMin-W3 (a CID-keyed font); kana characters from ShinGo-Light have been substituted; Tekton is specified as the Roman font. There is a +100 baseline shift (the original baseline was at 120, so the final baseline is at +220/1000 of the em space). Also, Gaiji characters are used from HeiseiMin-W3 Gaiji01.

Example 1: *Sample Rearranged CID sfnt Font*

```
< binary CSL procset data >
%ADOStartRearrangedFont
/HeiseiMin-W3-KS-G-RT-90pv-RKSJ-H [/HeiseiMin-W3-90pv-RKSJ-H
/Tekton /ShinGo-Light-83pv-RKSJ-H /HeiseiMin-W3-G01]
beginrearrangedfont
1 beginusematrix [1 0 0 1 0 0.1] endusematrix
1 usefont
9 beginbfrange
<00> <26> <00>
<27> <27> <a9>
<28> <5b> <28>
<5c> <5c> <a5>
<5d> <5f> <5d>
<60> <60> <c1>
<61> <7d> <61>
<7e> <7e> <c4>
<7f> <7f> <7f>
endbfrange
2 usefont
10 beginbfrange
<8152> <8155> <8152>
<815b> <815b> <815b>
<829f> <82f1> <829f>
<8340> <837e> <8340>
<8380> <8396> <8380>
<eb52> <eb55> <eb52>
<eb5b> <eb5b> <eb5b>
<ec9f> <ecf1> <ec9f>
<ed40> <ed7e> <ed40>
<ed80> <ed96> <ed80>
endbfrange
3 usefont
1 beginbfrange
<f000> <f0ff> <00>
endbfrange
endrearrangedfont
end
%%EndResource
```

7 COFN

The COFN table is used only for rearranged CID sfnt fonts. It is used to specify the QuickDraw font name for each component font used in the rearranged font. The COFN table consists of a Header followed by COFN Data, as shown in Figure 7. COFN Data contains one COFN Name Record for each QuickDraw style, as shown in Table 15.

Figure 7 *COFN Table Layout*

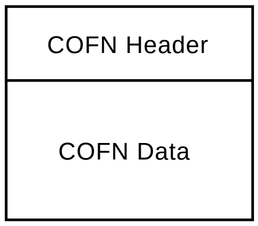


Table 13 shows the format of the COFN Header table.

Table 13 *COFN Header Table*

Type	*Field*
Fixed	version (0x00010000)
uint16	entryCount
uint16	offsets[entryCount+1]

The version field should be set to 1.0 (0x00010000).

entryCount is equal to the number of component fonts used in the rearranged font, and the offsets array contains offsets from the beginning of the COFN table to each COFNNameRecord entry in the COFNData table. The format of the COFNData table is shown in Table 14.

Table 14 *COFNData Table*

Type	*Field*
COFNNameRecord	nameRecord[<variable>]

A COFNNameRecord entry has a variable length which can be calculated by subtracting the offset for the chosen entry from the offset for the next entry. For this reason, the values in the offsets array must be sorted in increasing order, and one additional offsets entry is required.

The format of each nameRecord entry is shown in Table 15.

Table 15 *NameRecord Table*

Type	*Field*
uint8	styleBits
uint8	PString[<variable>]

styleBits is the QuickDraw style bits, which is used with the QuickDraw font name to access the QuickDraw font.

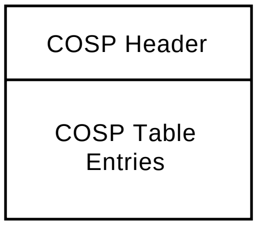
PString is the QuickDraw font name of the component font in the Pascal string format (one byte for the string length, followed by the string).

8 COSP

The COSP table defines the code space of a rearranged font, and is required for, and used only, with rearrranged fonts. The code space defines the numbering of the virtual glyph index assigned to each code point of the rearranged font. For a definition of the code space, refer to Adobe Technical Note #5014, “Adobe CMap and CIDFont Files Specification.”

The COSP table consists of a Header and one or more entries as shown in Figure 8.

Figure 8 *COSP Table Layout*



The COSP Header format is shown in Table 16.

Table 16 *COSP Table Header*

Type	*Field*
Fixed	version
uint16	flags
uint16	entryCount
COSPEntry	entry[entryCount]

The version field should be set to 1.0 (0x00010000).

The flags field is reserved for future use, and should be set to 0.

entryCount is the number of entries in the table.

The format for each entry is shown in Table 17.

Table 17 *COSP Entry Table Element*

Type	*Field*
uint16	numBytes
uint16	firstCode
uint16	lastCode

Each entry represents one code space range.

numBytes is the number of bytes of a character code in the code space range represented by this entry. The value of numBytes is either 1 or 2.

firstCode is the first character code in the range.

lastCode is the last character code in this range.

If numBytes is 1, the single byte code is placed in the lower byte of firstCode and lastCode. The upper byte must be zero. The entries must be sorted into increasing character code order.

A glyph index is assigned to each character code defined in the code space, in increasing order. The glyph index 0 is assigned to the first character code in the range, 1 is assigned to the next, and so on. The glyph indices defined in this way are used in the cmap table of the rearranged font.

For example, if the COSP table elements are as follows:

numBytes	**first**	**last**
1	20	7F
2	8120	9FFC
. . .		

Then the corresponding cmap table would look like:

char code	**glyph index**
20	0
21	1
. . .	. . .
7F	95
8120	96
8121	97
. . .	. . .
81FC	315

8220	316
8221	317
. . .	. . .

Note *The total number of glyph indices defined by the* COSP *table must be specified in the* CIDCount *field in the* CID *table of a rearranged CID sfnt font.*

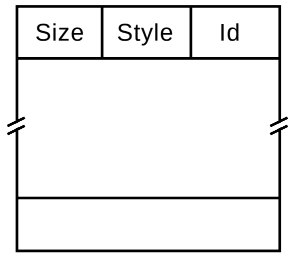
9 FNAM

The FNAM table provides Macintosh QuickDraw compatibility for CID sfnt fonts.

The TrueType font format introduced sfnt resources to the QuickDraw graphics model, but retained FOND resources as a means of access. QuickDraw accesses all fonts through FOND resources. These resources were originally designed to unite a family of NFNT bitmaps, each containing up to 256 glyphs. A typical QuickDraw rasterization request includes a FOND resource id, an 8-bit character code, a QuickDraw style (bold, italic, etc.), and a desired size. The size and QuickDraw style are used to access a Font Association Table within the FOND and retrieve the NFNT id of the bitmap to be displayed.

Figure 9 shows the Font Association Table in a Macintosh FOND resource, with the fields for Size, Style, and Id. There is one such entry for each QuickDraw style associated with the font. A Size value of zero (the units are points) indicates that the Id is for an sfnt rather than for an NFNT resource.

Figure 9 *FOND Font Association Table*



ATM also accesses sfnt resources under the QuickDraw graphics model via FOND resources. As with TrueType fonts, ATM rasterizer retrieves the sfnt Id from the Font Association Table entry with the appropriate QuickDraw style and a Size value of 0. The sfnt id is then used by the Type 1 rasterizer to read the sfnt resource and rasterize the required glyph.

One or more QuickDraw FOND resources may refer (via the Font Association Table) to the same sfnt font. Each FOND is said to represent an encoding set because access using that FOND will encode a sub-set of the sfnt glyphs.

For example, character code 97 used with a hypothetical GaramondRegular FOND would select the ‘a’ glyph, while the same character code from the GaramondExpert FOND would select the ‘small capital A’ glyph.

The FNAM table is used to select one of the available sfnt encoding sets. This depends on the QuickDraw FOND resource being used. CID sfnt fonts use the cmap table to specify an encoding, unlike Type 1 sfnt fonts which use the ENCO table.

The layout of the FNAM table, shown in Figure 10, consists of an FNAM Header, shown in Table 18, followed by an selector table, shown in Table 19.

Figure 10 *FNAM Table Layout*

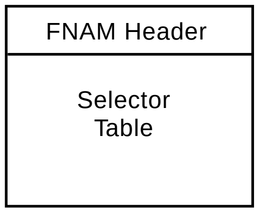


Table 18 *FNAM Header Table*

Type	*Field*
Fixed	version
uint16	encSetCount
uint16	offsets[EncSetCount + 1]

The version field should be set to 1.0 (0x00010000).

encSetCount is the number of encoding sets in the given font. The minimum encSetCount is 1.

The offsets array contains offsets from the beginning of the FNAM table to the first selector of each encoding set. The offsets must be numerically increasing and, for CID-keyed fonts, each selector matches the language ID of the cmap subtable. The number of selectors for each encoding set is determined by inspecting the next offset in the array. One additional entry is required so that the number of selectors for the last encoding set may be determined.

Table 19 *Selector Table*

Type	*Field*
Selector	selector[<variable>]

Each selector element in the array has the following form:

Table 20 *Selector Element*

Type	*Field*
uint8	style
PascalStr	name[<variable>]

The style field contains the QuickDraw style bits (described in *Inside Macintosh – Text*).

The name field contains the FOND name.

This table is also used to access NFNT bitmaps from QuickDraw GX.

10 HFMX

The HFMX table specifies font-wide metrics for the font when it is being set horizontally.

Table 21 *HFMX Table*

Type	*Field*
Fixed	version
FWord	ascent
FWord	descent
FWord	lineGap
int16	caretSlopeRise
int16	caretSlopeRun
FWord	caretOffset

The version field should be set to 1.0 (0x00010000).

The metric ascent is the y-value of the top of the em space in which the character is drawn. (Adobe uses a value of 880 character space units.)

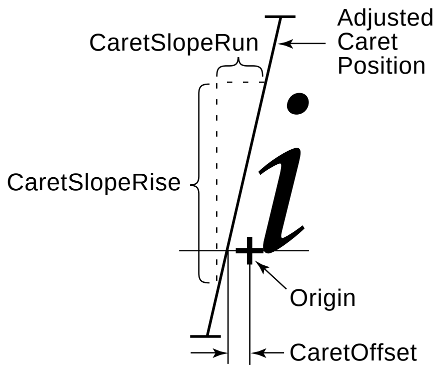
The metric descent is the y-value of the bottom of the em space. (Adobe uses a value of –120. While these values are not required, fonts with values that conform are more likely to align correctly when the fonts are used in horizontal writing mode.)

The metric lineGap is the amount of leading to add between successive lines of text. This value is set to $^1/_{12}$ em, which is 1000÷12 = 83 FUnits for Adobe-produced fonts (this value only used in the Macintosh system).

caretSlopeRise, caretSlopeRun, and caretOffset specify the orientation and offset of the editing caret for use with this font. A caretSlopeRise of 1 with a caretSlopeRun of 0 gives the standard vertical caret. caretOffset specifies a horizontal adjustment, in FUnits, of the caret so that it appears optically centered between adjoining glyphs. The values of caretSlopeRise and caretSlopeRun only need to express the correct ratio, rather than specific measurements. Figure 11 illustrates the relationship between these parameters.

Note *Most CID-keyed sfnt fonts for CJK languages are likely to have a vertical caret.*

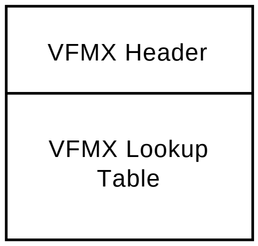
Figure 11 *Caret Parameters*



11 VFMX

The VFMX table specifies line and glyph metrics for font when it is being set vertically. This table consists of two parts: the VFMXHeader and the VFMX Lookup Table; as shown in Figure 12.

Figure 12 *VFMX Table Layout*



The VFMXHeader has the following format:

Table 22 *VFMX Header Table*

Type	*Field*
Fixed	version
int16	flags
uint16	nMasters
int16	before
int16	after
int16	caretSlopeRise
int16	caretSlopeRun
int16	caretOffset
VFMXData	defaultEntry

The version field should be set to 2.0 (0x00020000).

The flags field is reserved for future use, and should be set to 0.

The nMasters field should be set to a value of 1 for non–multiple master fonts.

The before and after fields are in FUnits and specify the space to the right and left of the vertical line being set.

caretSlopeRise, caretSlopeRun, and caretOffset specify the orientation and offset of the editing caret for use with this font. A caretSlopeRise of 0 along with a caretSlopeRun of 1 would give a horizontal caret. caretOffset specifies a horizontal adjustment, in FUnits, of the caret so that it appears optically centered between adjoining glyphs.

The format for the defaultEntry is shown in Table 23. This format is also used for Lookup Data for formats 0, 2, 6, and 8; as well as for the optional subtable (shown in Figure 13) for Lookup format 4.

Note *Formats 0, 2, 4, 6, and 8 are supported as documented in Apple's QuickDraw GX Font Formats. However, only formats 2 and 4 are used for CID sfnt fonts.*

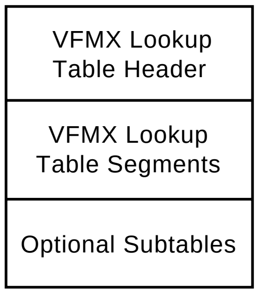
Table 23 *VFMX Data Entry*

Type	*Field*
int16	verticalAdvance
int16	xOffsetToVOrigin
int16	yOffsetToVOrigin

verticalAdvance specifies the vertical advance of the glyph in the vertical writing mode. xOffsetToVOrigin and yOffsetToVOrigin specify the offset from the origin of the glyph in the horizontal writing mode to the origin of the glyph in the vertical writing mode. These metrics values apply to all glyphs looked up in the segment. All values are specified in FUnits.

The VFMX Lookup Table, which follows the VFMX Header, consists of three parts: a Lookup Table header, a table of segments, and optional subtables. The layout is shown in Figure 13.

Figure 13 *VFMX Lookup Table Layout*



The VFMX Lookup Table Header format is described in Table 28 in Appendix D.

The VFMX Lookup Table Segments consist of two parts. The first part of the segment depends on the Lookup Table format. For example, for format 2, the table contents are shown in Table 31 in Appendix D.

The second part of the segment is the Lookup data. For format 4, the format is shown in Table 24. For formats 0, 2, 6, and 8, the VFMX Data Entry is embedded in the second part of the segment rather than using the optional subtable.

Table 24 *VFMX Lookup Data for Format 4*

Type	*Field*
uint8	flags
int24	offsetToSubTable

The flags field is defined as shown in Table 25.

Table 25 *flags Values for the VFMX Lookup Data Table*

Name	*Value*
ONE_ENTRY_IN_SUBTABLE	0x80

The flags field may have a flag ONE_ENTRY_IN_SUBTABLE (0x80) set to indicate that there is only one entry in the subtable. The other bits are reserved for future use, and should be set to zero.

The offsetToSubTable field contains the offset to the subtable, which is shown in Table 26.

Table 26 VFMX *SubTable*

Type	*Field*
VFMXDataEntry	VFMXSubTableEntry[<variable>]

The number of entries contained in VFMXSubTableEntry for a Lookup segement depends on the value of the flags bit. Each Lookup data has an offset from the start of the Lookup table to the subtable which contains the metrics information.

12 QuickDraw and Rearranged Font Compatibility Issues

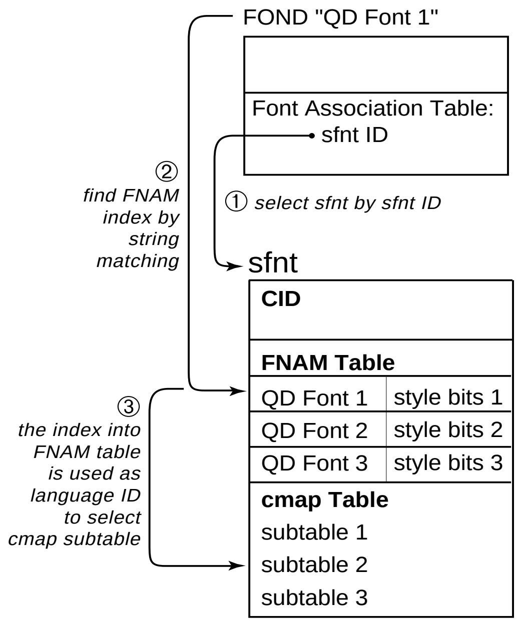
This section explains how encodings are selected in QuickDraw, and how rearranged CID-keyed fonts are supported by the CID sfnt font format.

12.1 How Encodings are Selected in QuickDraw Compatibility Mode

For QuickDraw compatibility, the FNAM table is used to map a QuickDraw font name to a cmap subtable for both the rasterizing and printing of a CIDFont. Unlike the Roman Type 1 GX format, the ENCO table is not used and is not included in a CID sfnt font.

The mechanism for selecting a cmap file is shown in Figure 14.

Figure 14 *cmap Selection in QuickDraw Mode*



First, the sfnt ID in the Font Association Table is used to select the correct sfnt resource. Second, the correct FNAM table index is found in that sfnt by string matching the QuickDraw font name. Third, that index is used as the language ID to select the cmap subtable.

The CID sfnt font should have one cmap subtable for each FNAM table entry, other than those selected by QuickDraw GX. Those cmap subtables should have the platform ID set to GXCustomPlatform (=5), the script ID set to gxCustom816BitScript (=2), and language ID set to the index (zero based) to the corresponding FNAM table entry.

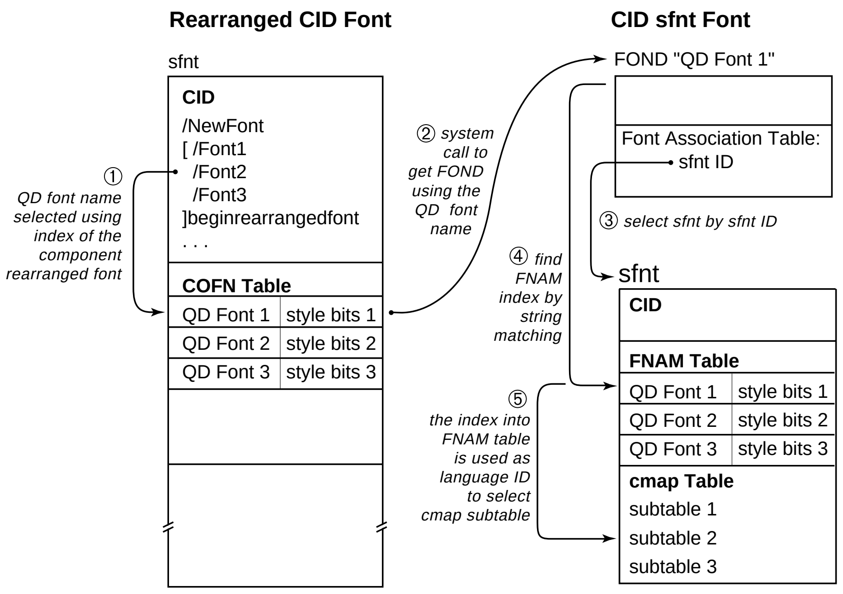
12.2 Rearranged CID sfnt Fonts

A rearranged CID font file references a template font and one or more component fonts. This indirection provides an ability to go from the character codes specified in the cmap table of the rearranged font to the glyph index in the component font.

Each component font must exist as a CID sfnt font or a Type 1 sfnt font. The component font is accessed by looking up its QuickDraw font name in the COFN table, using the same index used to access the PostScript font name in the component font list in the CID table as shown in Figure 15. This yields the QuickDraw FOND resource font name of the component font and the associated style bits.

The rest of the process is the same as that described in section 12.1. The name and style bits are then used in a system call to locate the correct FOND resource. The sfnt ID in the Font Association Table is used to select the correct sfnt resource, and the correct FNAM table index is found in that sfnt by string matching the QuickDraw font name. Lastly, that index is used as the language ID to select the cmap subtable (the other arguments, the platform and script ID, are constants for this operation).

Figure 15 *Accessing Component Fonts from a CID-keyed Rearranged Font*



Appendix A
sfnt Data Types

The following data types are used in sfnt fonts:

Data Type	*Description*
uint8	8-bit unsigned integer
int8	8-bit signed integer
uint16	16-bit unsigned integer
int16	16-bit signed integer
uint24	24-bit unsigned integer
int32	32-bit signed integer
Fixed	16.16-bit signed fixed-point number
FUnit	Smallest measurable distance in the em space (16-bit signed integer)
FWord	16-bit signed integer that describes a quantity in FUnits
PascalStr	Array of 8-bit unsigned integers whose first element specifies the number of succeeding elements.

Appendix B Bibliography

Apple

“The TrueType Font Format Specification, Version 1.0”. Apple Computer, 1990; APDA M0825LL/A.

Describes the concept and implementation of TrueType sfnt fonts.

“QuickDraw GX Font Formats: The TrueType Font Format Specification” (Developer Press, R0601LL/A), Apple Computer, 1993/4.

Describes the typographic capabilities available with sfnt fonts, along with complete technical specifications for the tables required to support those features.

Inside Macintosh – QuickDraw GX Typography. Addison Wesley, 1994.

Contains an overview of typographic concepts, programming examples, and programmer’s reference material for QuickDraw GX.

Inside Macintosh – Text. Chapter 4, Font Manager. Addison Wesley, 1993; ISBN 0-201-63298-5.

Complete information on the QuickDraw Font Manager, including how text is scaled and displayed. Also a section on the formats of the FOND and sfnt resources.

The Type 1 GX Font Format, Version 1.0. Included in the Apple Developers CD-ROM.

This document described the original sfnt format for Type 1 Roman (single-byte) fonts to be used in the Apple GX system.

Adobe

All of the following documents can be found on Adobe’s FTP and WWW servers at the following addresses:

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

ftp://ftp.adobe.com/pub/adobe/DeveloperSupport/TechNotes/

Adobe Type 1 Font Format. Addison Wesley, 1991; ISBN 0-201-57044-0.

Specification of the Type 1 font format. This document describes the platform-independent form of the format; formats for Macintosh and Windows host-based binary fonts are described in Technical Note #5040, "Supporting Downloadable PostScript Fonts."

Technical Note #0091: "Macintosh FOND Resource."

Describes the layout and use of Macintosh FOND resources, with examples of how file names of PostScript Type 1 fonts are derived.

Technical Note #5014: "Adobe CMap and CIDFont Files Specification."

Specification and tutorials on the CID-keyed font technology which is used for multi-byte encodings for PostScript Type 1 fonts.

Technical Note #5015: "The Type 1 Font Format Supplement."
This document contains all updates to the Type 1 format, including the specification of the multiple master font format. This document does not include the CID-keyed format extensions.

Technical Note #5040: "Supporting Downloadable PostScript Fonts."
Describes how Type 1 fonts have traditionally been packaged for use in the Macintosh and Windows environments — specifically, the use of POST resources for Macintosh fonts and the PFB compressed binary format for Windows fonts.

Technical Note #5087: "Multiple Master Font Programs for the Macintosh."
The BLND resource table information described in 5087 is the same as the current document; However, it also describes additional information on FOND resources for multiple master fonts, axis label information, and multiple master font naming conventions.

Technical Note #5088 "Font Naming Issues."
In addition to a discussion of general font name issues, this document explains the naming conventions for multiple master fonts.

Technical Note #5092: "CID-Keyed Font Technology Overview."
An overview of the nature and benefits of the CID-keyed font technology.

Technical Note #5213, "PostScript Language Extensions for CID-Keyed Fonts."

This document specifies additional CIDFont types and FMap types, as well as related extensions for supporting CID-keyed fonts in version 2015 of the PostScript interpreter.

Appendix C
Glossary

AFM file: Adobe Font Metrics File. An ASCII file containing glyph widths, bounding boxes, kerning data, and font-wide information.

ATM: Adobe Type Manager Software; a font rasterizer for Type 1 font programs.

caret: A bar that is perpendicular to, or at a slight angle from perpendicular, the writing direction that marks the point at which text is to be inserted or deleted.

character: A symbol that represents a sound, syllable, or notion used in a writing system; a member of a set of elements used for the organization, control, or represenation of data.

CID-keyed font: A file organization for multi-byte Type 1 fonts; glyphs are accessed by Character ID (CID) instead of by name lookup.

design coordinate: User-space coordinates for multiple master fonts; used for user interfaces and font names, and generally in the range from 0 to 1000.

em-square: In typography, the em space is the square of the point size. As related to glyph definitions, the space in which glyphs are “drawn” using Type 1 operators. Typically, 1000 x 1000 units for a Type 1 font.

glyph: A recognizable abstract graphic symbol which is independent of any specific design (i.e. Typeface design). A glyph is the final presentation form of one or more characters. For example, f and i are both characters and glyphs; the fi ligature is only a glyph.

glyph image: A particular image of a glyph; for example, a Garamond fi ligature.

FOND: A Macintosh resource; a data structure containing font metrics, encoding, and font family information.

FONT: An early type of Macintosh font data resource used for bitmap fonts.

font: an implementation of a typeface. Traditionally a single point size when the implementation was in metal or film; in the more flexible digital medium a font may be scalable to any size.

FUnits: The smallest measurable unit in a TrueType glyph’s em-square.

kerning: Fine adjustments made to the inter-glyph spacing specified in the font.

NFNT: A Macintosh resource containing one or more bitmap screenfonts.

optical size: The design or generation of glyphs that are optically correct for the size at which the font is being imaged. Also, an optional axis of a multiple master font.

rasterizer: Software that converts outline font glyph descriptions into a bitmap for imaging on a raster device.

typeface: A design for set of characters, numerals, and symbols based on a consistent design theme. For example, Bodoni Bold.

units-per-em: The number of relative units in an em-square.

Appendix D: Binary Search and Lookup Table Formats

This appendix contains information that is specified in the Apple GX Font Format specification, but which is reproduced here for the convenience of the reader, since a number of CID-specific tables refer to those tables.

This section describes two table components: binary searching tables and the lookup tables. *Binary search tables* contain data that assist in doing binary searches for information in the table that it resides in. *Lookup tables* map glyph indexes to information without context.

Binary Searching Tables

To minimize the time needed for text processing, many of the font tables contain information that aids the process of searching for the entry associated with a particular glyph index. This information is contained in a BinSrchHeader structure; its format is given in Table 27.

Table 27 *Binary Search Header Format*

Type	*Name*
uint16	unitSize
uint16	nUnits
uint16	searchRange
uint16	entrySelector
uint16	rangeshift

unitSize is the size of the lookup unit for this search, in bytes.

nUnits is the number of units of the preceding size to be searched.

searchRange is the value of unitSize times the largest power of 2 that is less than or equal to the value of nUnits.

entrySelector is the log base 2 of the largest power of 2 that is less than or equal to the value of nUnits.

rangeShift is the value of unitSize times the difference of the value of nUnits minus the largest power of 2 less than or equal to the value of nUnits.

To guarantee that a binary search terminates, one or more special “end of search table” values must be included at the end of the data to be searched. The number of termination values that need to be included is table-specific. The value 0xFFFF specifies binary search termination. The presence of this value helps to optimize search performance, with a minimal cost of extra space in the font table. The binary search header is often used in lookup tables.

Lookup Tables

Lookup tables provide a method for accessing information related to a specific glyph index. Some lookup tables do simple array-type lookup. Others involve groupings, allowing you to treat many different glyph indexes in the same way (that is, to look up the same information about them). The top-level description of a Lookup table contains a format number and a format-specific header. The format of the Lookup Table header is given in Table 28.

Table 28 *Lookup Table Header Format*

Type	*Name*
uint16	format
variable	fsHeader

The value of format is the format number (0, 2, 4, 6, or 8) of this Lookup table.

The fsHeader is the format-specific header (each of which is described in the following sections), followed by the actual lookup data. The details of the fsHeader structure are given with the different formats.

The result of a lookup is referred to in the following descriptions as a *lookup value*. A lookup value is interpreted differently for different types of tables.

There are five formats of Lookup tables are shown in Table 29:

Table 29 *Lookup Table Formats*

Format	*Description*
0	**Simple array format**. The lookup data is an array of lookup values, indexed by the glyph index.
2	**Segment single format**. Each non-overlapping segment has a single lookup value that applies to all glyhphs in the segment. A segment is defined as a contiguous range of glyph indexes.

4	**Segment array format**. A segment mapping is performed (as with format 2), but instead of a single lookup value for all the glyphs in the segment, each glyph in the segment gets its own separate lookup value.
6	**Single Table Format**. The lookup data is a sorted list of <*glyph index, lookup value*> pairs.
8	**Trimmed array format**. The lookup data is a simple trimmed array indexed by glyph index.

Simple Array Format 0 Lookup Table

The fsHeader field of a format 0 lookup table presents an array of lookup values, indexed by glyph index. What these values represent depends on the font table for which the lookup table is used.

Segment Single Format 2 Lookup Table

The fsHeader field of a format 2 lookup table divides some portion of the glyph indexes into contiguous ranges or segments. The format of a segment single format 2 lookup table is given in Table 30.

Table 30 *Segment Single Format 2 Lookup Table Format*

Type	*Name*
BinSrchHeader	binSrchHeader
LookupSegment	segments[ ]

The units for this binary search are of type LookupSegment and always have a minimum length of 6.

segments are the actual segments. These must already be sorted, according to the first word in each one (the last glyph in each segment).

The format of a format 2 segment entry is given in Table 31.

Table 31 *Format 2 lookupSegment Format*

Type	*Name*
uint16	lastGlyph
uint16	firstGlyph
variable	value

lastGlyph is the last glyph index in this segment.

firstGlyph is the first glyph index in this segment.

value is the lookup value.

For a format 2 lookup table, the value is applied uniformly to all glyphs in the segment.

Segment Array Format 4 Lookup Tables

The fsHeader field of a format 4 lookup table divides some portion of the glyph indexes into contiguous ranges or segments. The format of a segment array format 4 lookup table is given in Table 32.

Table 32 *Segment Array Format 4 Lookup Table Format*

Type	*Name*
BinSrchHeader	binSrchHeader
LookupSegment	segments[ ]

binSrchHeader has units for this binary search type of type LookupSegment, and must always have a minimum length of 6.

segments is the actual segments. These must already be sorted according to the first word in each one (the last glyph in each segment).

The format of a format 4 LookupSegment is given in Table 33.

Table 33 *Format 4 LookupSegment Format*

Type	*Name*
uint16	lastGlyph
uint16	firstGlyph
variable	value

lastGlyph is the last glyph index in this segment.

firstGlyph is the first glyph index in this segment.

value is the lookup values.

For a format 4 lookup table, the lookup values can be interpreted in several ways. Sometimes they are 16-bit offsets from the start of the table to the data. Other times they are the actual data. Specific interpretations are described in the sections that describe tables that use lookup tables.

Single Table Format 6 Lookup Table

The fsHeader field of a format 6 lookup table stores the lookup data as a sorted list of pairs of a glyph index and its lookup result. The format of a single table format 6 lookup table is shown in Table 34.

Table 34 *Single Table Format 6 Lookup Table Format*

Type	*Name*
BinSrchHeader	binSrchHeader
LookupSingle	entries[ ]

The units for this binary search are of type LookupSingle and always have a minimum length of 4.

entries[] is the actual entries, sorted by glyph index.

The format of a format 6 LookupSingle is given in Table 35.

Table 35 *Format 6 lookupSingle Format*

Type	*Name*
uint16	glyph
variable	value

glyph is the glyph index.

value is the lookup value.

Trimmed Array Format 8 Lookup Table

The fsHeader field of a format 8 lookup table stores the lookup data as a simple trimmed array indexed by glyph index. The format of a trimmed array format 8 lookup table is given in Table 36.

Table 36 *Trimmed Array Format 8 Lookup Table Format*

Type	*Name*
uint16	firstGlyph
uint16	glyphCount
variable	valueArray[ ]

firstGlyph is the first glyph index in the trimmed array.

glyphCount is the total number of glyphs (equivalent to the last glyph minus the value of firstGlyph plus 1).

valueArray[] is the lookup values indexed by the glyph index minus the value of firstGlyph.