
**Information technology — Coding of
audio-visual objects —**

**Part 22:
Open Font Format**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 22: Format de police de caractères ouvert*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	vii
Introduction.....	viii
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Abbreviated terms	2
4 The Open Font file format.....	3
4.1 Description	3
4.2 Filenames	3
4.3 Data types	3
4.4 Table version numbers	4
4.5 Top-level OFF organization	5
4.5.1 Offset table	5
4.5.2 Table directory	5
4.5.3 Calculating checksums	6
4.6 Font collections	6
4.6.1 The Font Collection overview	6
4.6.2 The Font Collection file structure	7
4.6.3 TTC header	7
5 Open font tables	8
5.1 General	8
5.2 Required common tables	8
5.2.1 List of required tables	8
5.2.2 cmap – Character to glyph index mapping table	9
5.2.3 head – Font header.....	21
5.2.4 hhea – Horizontal header.....	23
5.2.5 hmtx – Horizontal metrics	24
5.2.6 maxp – Maximum profile	25
5.2.7 name – Naming table.....	26
5.2.8 OS/2 – Global font information table	45
5.2.9 Font class parameters	67
5.2.10 post – PostScript.....	67
5.3 Tables related to TrueType outlines.....	69
5.3.1 List of TrueType outlines tables	69
5.3.2 cvt – Control value table.....	69
5.3.3 fpgm – Font program	69
5.3.4 glyf – Glyph data.....	70
5.3.5 loca – Index to location.....	75
5.3.6 prep – Control value program.....	75
5.3.7 gasp – Grid-fitting and scan-conversion procedure table	76
5.4 Tables related to CFF outlines	78
5.4.1 List of CFF outline tables.....	78
5.4.2 CFF – Compact Font Format (version 1) table	78
5.4.3 CFF2 – Compact Font Format (version 2) table	78
5.4.4 VORG – Vertical origin table	88
5.5 Table for SVG glyph outlines	89
5.5.1 SVG – The SVG (Scalable Vector Graphics) table	89
5.5.2 Color Palettes	90
5.5.3 Glyph Identifiers	91
5.5.4 Glyph Semantics and Metrics	91
5.5.5 Glyph Rendering.....	91
5.5.6 SVG glyph examples	93

5.6	Tables related to bitmap glyphs.....	98
5.6.1	List of bitmap glyph tables	98
5.6.2	EBDT – Embedded bitmap data table.....	98
5.6.3	EBLC – Embedded bitmap location table	101
5.6.4	EBSC – Embedded bitmap scaling table.....	108
5.6.5	CBDT – Color bitmap data table.....	109
5.6.6	CBLC – Color bitmap location table	111
5.6.7	sbix – Standard bitmap graphics table.....	112
5.7	Optional tables.....	114
5.7.1	DSIG – Digital signature table	115
5.7.2	hdmx – Horizontal device metrics.....	117
5.7.3	kern – Kerning.....	118
5.7.4	LTSH – Linear threshold	120
5.7.5	MERG – Merge table	121
5.7.6	meta – Metadata table	125
5.7.7	PCLT – PCL 5 table.....	128
5.7.8	VDMX – Vertical device metrics	135
5.7.9	vhea – Vertical header table	137
5.7.10	vmtx – Vertical metric table	141
5.7.11	COLR – Color Table	143
5.7.12	CPAL – Palette Table	144
6	Advanced Open Font layout tables.....	147
6.1	Advanced Open Font layout extensions	147
6.1.1	Overview of advanced typographic layout extensions.....	147
6.1.2	TrueType versus OFF layout	149
6.1.3	OFF layout terminology	149
6.1.4	Text processing with OFF layout	151
6.1.5	OFF layout and Font variations.....	153
6.2	OFF layout common table formats	153
6.2.1	Overview	153
6.2.2	OFF layout and Font variations.....	154
6.2.3	Table organization	155
6.2.4	Scripts and languages	156
6.2.5	Features and lookups.....	159
6.2.6	Coverage table	162
6.2.7	Class definition table.....	164
6.2.8	Device and VariationIndex tables.....	165
6.2.9	Feature variations.....	167
6.2.10	Common table examples	170
6.3	Advanced typographic tables.....	178
6.3.1	BASE Baseline table.....	178
6.3.2	GDEF – The glyph definition table	199
6.3.3	GPOS – The glyph positioning table.....	211
6.3.4	GSUB – The glyph substitution table	263
6.3.5	JSTF – The justification table.....	296
6.3.6	MATH – The mathematical typesetting table	306
6.4	Layout tag registry.....	322
6.4.1	Scripts tags	323
6.4.2	Language tags.....	327
6.4.3	Feature tags.....	344
6.4.4	Baseline tags.....	406
7	OFF font variations	410
7.1	Font variations overview.....	410
7.1.1	General.....	410
7.1.2	Terminology	412
7.1.3	Variation space, default instances and adjustment deltas	414
7.1.4	Coordinate scales and normalization.....	417
7.1.5	Variation data	419
7.1.6	Variation data tables and miscellaneous requirements	428
7.1.7	Algorithm for interpolation of instance values.....	429

7.1.8	Interpolation example	432
7.1.9	Dynamic generation of static instance fonts	437
7.2	Font variations common table formats	438
7.2.1	Overview	438
7.2.2	Tuple variation store	439
7.2.3	Item variation stores	446
7.2.4	Design-variation axis tag registry	450
7.3	Font variations tables	455
7.3.1	avar – Axis variations table	455
7.3.2	cvar – CVT variations table	459
7.3.3	fvar – Font variations table	461
7.3.4	gvar – Glyph variations table	468
7.3.5	HVAR – Horizontal metrics variations table	478
7.3.6	MVAR – Metrics variations table	481
7.3.7	STAT – Style attributes table	485
7.3.8	VVAR – Vertical metrics variations table	497
8	Recommendations for OFF fonts	499
8.1	Byte ordering	499
8.2	'sfnt' version	499
8.3	Mixing outline formats	499
8.4	Filenames	499
8.5	Table alignment and length	500
8.6	Glyph 0: the .notdef glyph	500
8.7	'BASE' table	500
8.8	'cmap' table	500
8.9	'cvt' table	501
8.10	'fpgm' table	501
8.11	'glyf' table	501
8.12	'hdmx' table	501
8.13	'head' table	501
8.14	'hhea' table	501
8.15	'hmtx' table	502
8.16	'kern' table	502
8.17	'loca' table	502
8.18	'LTSH' table	502
8.19	'maxp' table	502
8.20	'name' table	502
8.21	'OS/2' table	504
8.22	'post' table	505
8.23	'prep' table	505
8.24	'VDMX' table	505
8.25	TrueType Collections	505
9	General recommendations	506
9.1	Optimized table ordering	506
9.2	Non-standard (Symbol) fonts	506
9.3	Baseline to baseline distances	506
9.4	Style bits	507
9.5	Drop-out control	507
9.6	Embedded bitmaps	507
9.7	OFF CJK font guidelines	508
9.8	Stroke reduction in variable fonts	508
9.9	Families with optical size variants	508
Annex A (informative)	Font Class and Font Subclass parameters	510
Annex B (informative)	Earlier versions of OS/2 – OS/2 and Windows metrics	521
Annex C (informative)	OFF Mirroring Pairs List	596
Annex D (informative)	The CFF2 CharString Format	603
Annex E (informative)	CFF2 DICT Encoding	622

Annex F (informative) Registration of Media Type: application/font-sfnt..... 624

Bibliography..... 627

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see <http://patents.iec.ch>).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This fourth edition cancels and replaces the third edition (ISO/IEC 14496-22:2015), which has been technically revised. It also incorporates the Amendments ISO/IEC 14496-22:2015/Amd.1:2017 and ISO/IEC 14496-22:2015/Amd.2:2017.

The main changes compared to the previous edition are as follows:

- new technology clauses were added;
- many existing clauses, subclauses, tables, figures and annexes were editorially revised.

A list of all parts in the ISO/IEC 14496 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Multimedia applications require a broad range of media-related standards. In addition to the typical audio and video applications, multimedia presentations include scalable 2D graphics and text supporting all languages of the world. Faithful reproduction of scalable multimedia content requires additional components including scalable font technology. The Open Font Format, which is based on the OpenType®¹ font format, was originally developed as an extension of the TrueType®² font format, adding support for PostScript®³ Compact Font Format (CFF) font data. OFF fonts and the operating system services which support OFF fonts provide users with a simple way to install and use fonts, whether the fonts contain TrueType outlines or CFF (PostScript Type1) outlines.

The Open Font Format addresses the following goals:

- broader multi-platform support;
- excellent support for international character sets;
- excellent protection for font data;
- smaller file sizes to make font distribution more efficient;
- excellent support for advanced typographic control.

CFF data included in OFF fonts may be directly rasterized or converted to the TrueType outline format for rendering, depending on which rasterizers have been installed in the host operating system. But the user model is the same: OFF fonts just work. Users will not need to be aware of the type of outline data in OFF fonts. And font creators can use whichever outline format they feel provides the best set of features for their work, without worrying about limiting a font's usability.

OFF fonts can include the OFF Layout tables, which allow font creators to design broader international and high-end typographic fonts. The OFF Layout tables contain information on glyph substitution, glyph positioning, justification, and baseline positioning, enabling text-processing applications to improve text layout.

As with TrueType fonts, OFF fonts allow the handling of large glyph sets using Unicode encoding. Such encoding allows broad international support, as well as support for typographic glyph variants.

Additionally, OFF fonts may contain digital signatures, which allows operating systems and browsing applications to identify the source and integrity of font files, including embedded font files obtained in web documents, before using them. Also, font developers can encode embedding restrictions in OFF fonts which cannot be altered in a font signed by the developer.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights. The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holders of these patent rights is registered with ISO and IEC.

¹ OpenType® is the trademark of a product supplied by Microsoft. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of this product.

² TrueType® is the trademark of a product supplied by Apple Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of this product.

³ PostScript® is the trademark of a product supplied by Adobe Systems Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of this product.

Information may be obtained from:

Apple Inc.
1 Infinite Loop MS 3-PAT
US-Cupertino, CA 95014-2084
Tel.: +1 408 974 9453
Email: iplaw@apple.com

Microsoft Corporation
Interoperability Group 3460 157th Avenue NE
US-Redmond, WA 98052
Tel.: +1 425 882 80 80

Monotype Imaging Inc.
500 Unicorn Park Drive
US-Woburn, MA 01801
Tel.: +1 781-970-6088
E-mail: vladimir.levantovsky@monotypeimaging.com

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Coding of audio-visual objects —

Part 22: Open Font Format

1 Scope

This document specifies the Open Font Format (OFF) specification, including the TrueType and Compact Font Format (CFF) outline formats. Many references to both TrueType and PostScript exist throughout this document, as Open Font Format fonts combine the two technologies. The document defines data structures for various font tables, and provides the necessary details for developers to build a font rendering and text layout/shaping engines in compliance with this document.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646, *Information technology — Universal Coded Character Set (UCS)*

ISO/IEC 14496-18, *Information technology — Coding of audio-visual objects — Part 18: Font compression and streaming*

ISO/IEC 15948, *Information technology — Computer graphics and image processing — Portable Network Graphics: Functional specification*⁴

IEC 61966-2-1/Amd 1:2003: *Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB*.

TrueType Instruction Set, <http://www.microsoft.com/typography/otspec/ttinst.htm>

Unicode 11.0, <http://www.unicode.org/versions/Unicode11.0.0/>

Scalable Vector Graphics (SVG) 1.1 (2nd edition), W3C Recommendation, 16 August 2011 <http://www.w3.org/TR/SVG11/>

IETF BCP 47 specification, “Tags for Identifying Languages”. <http://tools.ietf.org/html/bcp47>

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

⁴ Also available as a W3C Recommendation (Reference [15]).

3.2 Abbreviated terms

ACF	average character face
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
BMP	[Unicode] basic multilingual plane
BTBD	baseline to baseline distance
CFF	compact font format
CID	character identifier
CJK	Chinese Japanese Korean [characters, ideographs, fonts, etc.]
CJKV	Chinese Japanese Korean and Vietnamese
CV	control value
CVT	control value table
DLL	dynamically linked library
FDEF	function definition
GID	glyph ID
ICF	ideographic character face
IDEF	instruction definition
IETF	Internet Engineering Task Force
JIS	Japanese Industrial Standard
LTR	left to right
NLC	National Language Council of Japan
OFF	open font format
OMPL	OFF mirroring pairs list
OTF	OpenType font
PCL	printer control language
PPM, PPEM	pixels per em
RTL	right to left
TTC	TrueType collection
TTF	TrueType font
UCS	universal character set
UTF	Unicode transformation format
UVS	Unicode variation sequence
VM	virtual memory
W3C	world wide web consortium

4 The Open Font file format

4.1 Description

An Open Font file contains data, in table format, that comprises either a TrueType or a CFF outline font. Rasterizers use combinations of data from the tables contained in the font to render the TrueType or PostScript glyph outlines. Some of this supporting data is used no matter which outline format is used; some of the supporting data is specific to either TrueType or PostScript.

References to the Universally Coded Character Set and the Unicode standard are used throughout this document; the users of the OFF cannot meet the requirements of this document without strict adherence to these standards.

4.2 Filenames

OFF font files may have the extension .OTF, .TTF, .OTC or .TTC. The extensions .OTC and .TTC should only be used for font collection files. For additional information on filename extension conventions, see [subclause 8.4](#).

4.3 Data types

The following data types are used in the OFF font file. All OFF fonts use big-endian (network byte order):

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.
uint32	32-bit unsigned integer.
int32	32-bit signed integer.
Fixed	32-bit signed fixed-point number (16.16)
FWORD	int16 that describes a quantity in font design units.
UWORD	uint16 that describes a quantity in font design units.
F2DOT14	16-bit signed fixed number with the low 14 bits of fraction (2.14).
LONGDATETIME	Date represented in number of seconds since 12:00 midnight, January 1, 1904. The value is represented as a signed 64-bit integer.
Tag	Array of four uint8s (length = 32 bits) used to identify a table, design-variation axis, script, language system, feature, or baseline
Offset16	Short offset to a table, same as uint16, NULL offset = 0x0000
Offset32	Long offset to a table, same as uint32, NULL offset = 0x00000000

The F2DOT14 format consists of a signed, 2's complement integer and an unsigned fraction. To compute the actual value, take the integer and add the fraction. Examples of 2.14 values are:

Decimal Value	Hex Value	Integer	Fraction
1.999939	0x7fff	1	16383/16384
1.75	0x7000	1	12288/16384
0.000061	0x0001	0	1/16384
0.0	0x0000	0	0/16384
-0.000061	0xffff	-1	16383/16384
-2.0	0x8000	-2	0/16384

A Tag value is a uint8 array. Each byte within the array shall have a value in the range 0x20 to 0x7E. This corresponds to the range of values of Unicode Basic Latin characters in UTF-8 encoding, which is the same as the printable ASCII characters. As a result, a Tag value can be re-interpreted as a four-character sequence, which is conventionally how they are referred to. Formally, however, the value is a byte array.

When re-interpreted as characters, the Tag value is case sensitive. It shall have one to four non-space characters, padded with trailing spaces (byte value 0x20). A space character cannot be followed by a non-space character.

4.4 Table version numbers

Most tables have version numbers, and the version number for the entire font is contained in the Table Directory. Note that there are five different table version number types, each with its own numbering scheme.

- A single uint16 field. This is used in a number of tables, usually with versions starting at zero (0).
- Separate, uint16 major and minor version fields. This is used in a number of tables, usually with versions starting at 1.0.
- A Fixed field for major/minor version numbers. This is used in the [maxp](#), [post](#) and [vhea](#) tables.
- A uint32 field with enumerated values.
- A uint32 field with a numeric value. This is used only in the [DSIG](#) and [meta](#) tables.

Only certain tables use a Fixed value for version, and only for reasons of backward compatibility. Fixed values will not be used in the future for any new tables that may be introduced. When a Fixed number is used as a version, the upper 16 bits comprise a major version number and the lower 16 bits a minor version. The representation of a non-zero minor version, however, is not consistent with the normal treatment of Fixed values, in which the lower 16 bits represent a fractional value, $N * 2^{-16}$. Rather, tables with non-zero minor version numbers always specify the literal value of the version number. For example, the version number of 'maxp' table version 0.5 is 0x00005000, and that of 'vhea' table version 1.1 is 0x00011000. When Fixed is indicated as the type for a version field, the possible values should be treated as an enumeration of specific values, rather than as a numeric value capable of representing many potential major and minor versions.

The Table Directory uses a uint32 field with an enumeration of defined values that represent four-character tags; see subclause 4.5 (Top-level OFF organization) for details.

Implementations reading tables must include code to check version numbers so that if and when the format and therefore the version number changes, older implementations will handle newer versions gracefully.

Minor version number changes always imply format changes that are compatible extensions. If an implementation understands a major version number, then it can safely proceed reading the table. If the minor version is greater than the latest version recognized by the implementation, then the extension fields will be undetectable to the implementation.

For purposes of compatibility, version numbers that are represented using a single uint16 or uint32 value are treated like a minor version number, and any format changes are compatible extensions.

Note that some field values that were undefined or reserved in an earlier revision may be assigned meanings in a minor version change. Implementations should never make assumptions regarding reserved or unassigned values or bits in bit fields, and can ignore them if encountered. When writing font data, tools should always write zero for reserved fields or bits. Validators should warn of any non-zero values for fields or bits that are not defined for the given version against which data is being validated.

If the major version is not recognized, the implementation must not read the table as it can make no assumptions regarding interpretation of the binary data. The implementation should treat the table as missing.

4.5 Top-level OFF organization

4.5.1 Offset table

A key characteristic of the OFF format is the TrueType sfnt "wrapper", which provides organization for a collection of tables in a general and extensible manner.

The OFF font starts with the Offset Table. If the font file contains only one font, the Offset Table will begin at byte 0 of the file. If the font file is a font collection, the beginning point of the Offset Table for each font is indicated in the TTCHheader.

Offset Table		
Type	Name	Description
uint32	sfntVersion	0x00010000 or 0x4F54544F ('OTTO') – see below.
uint16	numTables	Number of tables.
uint16	searchRange	(Maximum power of 2 <= numTables) x 16.
uint16	entrySelector	Log2(maximum power of 2 <= numTables).
uint16	rangeShift	NumTables x 16 - searchRange.

OFF fonts that contain TrueType outlines should use the value of 0x00010000 for sfntVersion. OFF fonts containing CFF data (version 1 or 2) should use 0x4F54544F ('OTTO', when re-interpreted as a Tag) for sfntVersion.

4.5.2 Table directory

The Offset Table is followed immediately by the Table Record entries. Entries in the Table Record shall be sorted in ascending order by tag. Offset values in the Table Record are measured from the start of the font file.

Table Record		
Type	Name	Description
Tag	tableTag	Table identifier.
uint32	checksum	Checksum for this table.
Offset32	Offset	Offset from beginning of TrueType font file.
uint32	Length	Length of this table.

The Table Record makes it possible for a given font to contain only those tables it actually needs. As a result there is no standard value for numTables.

Table tags are the names given to tables in the OFF font file. For requirements of Tag values, see subclause 4.3 (Data types).

Some tables have an internal structure with subtables located at specified offsets, and as a result, it is possible to construct a font with data for different tables interleaved. In general, tables should be arranged contiguously without overlapping the ranges of distinct tables. In any case, however, table lengths measure a contiguous range of bytes that encompasses all of the data for a table. This applies to any subtables as well as to top-level tables.

4.5.3 Calculating checksums

Table checksums are the unsigned sum of the longs of a given table. In C, the following function can be used to determine a checksum:

```
uint32
CalcTableChecksum(uint32 *Table, uint32 Length)
{
    uint32 Sum = 0L;
    uint32 *EndPtr = Table + ((Length+3) & ~3) / sizeof(uint32);

    while (Table < EndPtr)
        Sum += *Table++;
    return Sum;
}
```

NOTE This function implies that the length of a table is a multiple of four bytes. In fact, a font is not considered structurally proper without the correct padding. All tables must begin on four byte boundaries, and any remaining space between tables is padded with zeros. The length of all tables would be recorded in the table record with their actual length (not their padded length).

To calculate the checkSum for the 'head' table which itself includes the checkSumAdjustment entry for the entire font, do the following:

- Set the checkSumAdjustment to 0.
- Calculate the checksum for all the tables including the 'head' table and enter that value into the table directory.
- Calculate the checksum for the entire font.
- Subtract that value from value 0xB1B0AFBA.
- Store the result in checkSumAdjustment.

The checkSum for the head table which includes the checkSumAdjustment entry for the entire font is now incorrect. That is not a problem. Do not change it. An application attempting to verify that the 'head' table has not changed should calculate the checkSum for that table by not including the checkSumAdjustment value, and compare the result with the entry in the table directory.

4.6 Font collections

4.6.1 The Font Collection overview

A Font Collection (either TTC or OTC, formerly known as TrueType Collection) is a means of delivering multiple OFF font resources in a single file structure. Font collections containing either TrueType or CFF outlines (TTC or OTC) are most useful when the fonts to be delivered together share many glyphs in common. By allowing multiple fonts to share glyph sets and other common font tables, font collections can result in a significant saving of file space.

For example, a group of Japanese fonts may each have their own designs for the kana glyphs, but share identical designs for the kanji. With ordinary OFF font files, the only way to include the common kanji glyphs is to copy their glyph data into each font. Since the kanji represent much more data than the kana, this results in a great deal of wasteful duplication of glyph data. Font collections were defined to solve this problem.

NOTE Even though the original definition of TrueType Collection (as part of the TrueType specification) was intended to be used with fonts containing TrueType outlines, this is no longer strictly the case. TTC files may contain various types of outlines (or a mix of them), regardless of whether or not fonts have layout tables present. For backward compatibility and simplicity, the description of the font collection file structure is using the term "TrueType Collection" though it is understood that it is used to identify a generic font collection structure containing any type of outline tables.

4.6.2 The Font Collection file structure

A font collection file consists of a single TTC Header table, one or more Offset Tables with Table Directories (each corresponding to a different font resource), and a number of OFF tables. The TTC Header shall be located at the beginning of the TTC file.

The TTC file shall contain a complete Offset Table and Table Directory for each font resource. A TTC file Table Directory has exactly the same format as a TTF file Table Directory. The table offsets in all Table Directories within a TTC file are measured from the beginning of the TTC file.

Each OFF table in a TTC file is referenced through the Offset Table and Table Directory of each font which uses that table. Some of the OFF tables must appear multiple times, once for each font included in the TTC; while other tables may be shared by multiple fonts in the TTC.

As an example, consider a TTC file which combines two Japanese fonts (Font1 and Font2). The fonts have different kana designs (Kana1 and Kana2) but use the same design for kanji. The TTC file contains a single 'glyf' table which includes both designs of kana together with the kanji; both fonts' Table Directories point to this 'glyf' table. But each font's Table Directory points to a different 'cmap' table, which identifies the glyph set to use. Font1's 'cmap' table points to the Kana1 region of the 'loca' and 'glyf' tables for kana glyphs, and to the kanji region for the kanji. Font2's 'cmap' table points to the Kana2 region of the 'loca' and 'glyf' tables for kana glyphs, and to the same kanji region for the kanji.

The tables that should have a unique copy per font are those that are used by the system in identifying the font and its character mapping, including 'cmap', 'name', and 'OS/2'. The tables that should be shared by fonts in the TTC are those that define glyph and instruction data or use glyph indices to access data: 'glyf', 'loca', 'hmtx', 'hdmx', 'LTSH', 'cvt', 'fpgm', 'prep', 'EBLC', 'EBDT', 'EBSC', 'maxp', and so on. In practice, any tables which have identical data for two or more fonts may be shared.

4.6.3 TTC header

There are two versions of the TTC Header: Version 1.0 has been used for TTC files without digital signatures. Version 2.0 can be used for TTC files with or without digital signatures -- if there's no signature, then the last three fields of the version 2.0 header are left null.

If a digital signature is used, the DSIG table for the file must be the last table in the TTC file. Signatures in a TTC file are expected to be Format 1 signatures.

The purpose of the TTC Header table is to locate the different Offset Tables within a TTC file. The TTC Header is located at the beginning of the TTC file (offset = 0). It consists of an identification tag, a version number, a count of the number of OFF fonts in the file, and an array of offsets to each Offset Table.

TTC Header Version 1.0		
Type	Name	Description
TAG	ttcTag	Font Collection ID string: 'ttcf' (used for fonts with CFF or CFF2 outlines, as well as TrueType outlines)
uint16	majorVersion	Major version of the TTC Header, = 1.

uint16	minorVersion	Minor version of the TTC Header, = 0.
uint32	numFonts	Number of fonts in TTC
Offset32	offsetTable[numFonts]	Array of offsets to the OffsetTable for each font from the beginning of the file

TTC Header Version 2.0		
Type	Name	Description
TAG	ttcTag	TrueType Collection ID string: 'ttcf'
uint16	majorVersion	Major version of the TTC Header, = 2.
uint16	minorVersion	Minor version of the TTC Header, = 0.
uint32	numFonts	Number of fonts in TTC
Offset32	offsetTable[numFonts]	Array of offsets to the OffsetTable for each font from the beginning of the file
uint32	ulDsigTag	Tag indicating that a DSIG table exists, 0x44534947 ('DSIG') (null if no signature)
uint32	dsigLength	The length (in bytes) of the DSIG table (null if no signature)
uint32	dsigOffset	The offset (in bytes) of the DSIG table from the beginning of the TTC file (null if no signature)

5 Open font tables

5.1 General

The rasterizer has a much easier time traversing tables if they are padded so that each table begins on a 4-byte boundary. All tables shall be long-aligned and padded with zeroes.

5.2 Required common tables

5.2.1 List of required tables

Whether TrueType or CFF outlines are used in an OFF font, the following tables are required for the font to function correctly:

Tag	Name
cmap	Character to glyph mapping
head	Font header
hhea	Horizontal header

hmtx	Horizontal metrics
maxp	Maximum profile
name	Naming table
OS/2	OS/2 and Windows specific metrics
post	PostScript information

5.2.2 cmap – Character to glyph index mapping table

This table defines the mapping of character codes to the glyph index values used in the font. It may contain more than one subtable, in order to support more than one character encoding scheme.

5.2.2.1 Table overview

This table defines the mapping of character codes to a default glyph index. Different subtables may be defined that each contain mappings for different character encoding schemes. The table header indicates the character encodings for which subtables are present.

Regardless of the encoding scheme, character codes that do not correspond to any glyph in the font should be mapped to glyph index 0. The glyph at this location must be a special glyph representing a missing character, commonly known as .notdef.

Each subtable is in one of seven possible formats and begins with a format code indicating the format used. The first four formats — formats 0, 2, 4 and 6 — were originally defined prior to Unicode 2.0. These formats allow for 8-bit single-byte, 8-bit multi-byte, and 16-bit encodings. With the introduction of supplementary planes in Unicode 2.0, the Unicode addressable code space extends beyond 16 bits. To accommodate this, three additional formats were added — formats 8, 10 and 12 — that allow for 32-bit encoding schemes.

Other enhancements in Unicode led to the addition of other subtable formats. Subtable format 13 allows for an efficient mapping of many characters to a single glyph; this is useful for “last-resort” fonts that provide fallback rendering for all possible Unicode characters with a distinct fallback glyph for different Unicode ranges. Subtable format 14 provides a unified mechanism for supporting Unicode variation sequences.

NOTE The 'cmap' table version number remained at 0x0000 for fonts that make use of the newer subtable formats.

'cmap' Header

The Character to Glyph Index Mapping Table is organized as follows:

Type	Name	Description
uint16	Version	Table version number (0)
uint16	numTables	Number of encoding tables that follow
EncodingRecord	encodingRecords[numTables]	

Encoding records and encodings

The array of encoding records specify particular encoding and the offset to the subtable for each encoding.

Encoding Record:

Type	Name	Description
uint16	platformID	Platform ID.
uint16	encodingID	Platform-specific encoding ID.
Offset32	offset	Byte offset from beginning of table to the subtable for this encoding

The platform ID and platform-specific encoding ID in the encoding record are used to specify a particular character encoding. In the case of the Macintosh platform, a language field within the mapping subtable is also used for this purpose.

The encoding record entries in the 'cmap' header must be sorted first by platform ID, then by platform-specific encoding ID, and then by the language field in the corresponding subtable. Each platform ID, platform-specific encoding ID, and subtable language combination may appear only once in the 'cmap' table.

Complete details on platformIDs, and platform-specific encoding and language IDs are defined in [subclause 5.2.7](#). Some specific details applicable to the 'cmap' table are provided here.

Unicode platform (platform ID = 0)

Unicode Variation Sequences supported by the font should be specified in the 'cmap' table using a format 14 subtable. A format 14 subtable must only be used under platform ID 0 and encoding ID 5. *Macintosh platform (platform ID = 1)*

When building a font that will be used on the Macintosh, the platform ID should be 1 and the encoding ID should be 0.

Windows platform (platform ID = 3)

When building a Unicode font for Windows, the platform ID should be 3 and the encoding ID should be 1. When building a symbol font for Windows, the platform ID should be 3 and the encoding ID should be 0.

Microsoft strongly recommends using a BMP Unicode 'cmap' for all fonts. However, other non-Unicode encodings are also used in existing fonts with the Windows platform. The following are encoding IDs defined for the Windows platform:

Windows encodings		
Platform ID	Encoding ID	Description
3	0	Symbol
3	1	Unicode BMP (UCS-2)
3	2	ShiftJIS
3	3	PRC
3	4	Big5
3	5	Wansung
3	6	Johab
3	7	Reserved

3	8	Reserved
3	9	Reserved
3	10	Unicode UCS-4

Fonts that support Unicode BMP characters on Windows platform must have a Format 4 'cmap' subtable for platform ID 3, platform-specific encoding 1.

Fonts that support Unicode supplementary-plane characters on Windows platform must have a Format 12 subtable for platform ID 3, encoding ID 10. To ensure backward compatibility with older software and devices, a format 4 subtable for platform ID 3, encoding ID 1 is also required. The characters supported in the format 4 subtable must be a subset of the characters in the format 12 subtable and should include all of the Unicode BMP characters supported by the font.

Custom platform (Platform ID = 4) and OTF Windows NT compatibility mapping

If a platform ID 4 (custom), encoding ID 0-255 (OTF Windows NT compatibility mapping) 'cmap' encoding is present in an OFF font with CFF outlines, then the OTF font driver in Windows NT will:

- superimpose the glyphs encoded at character codes 0-255 in the encoding on the corresponding Windows character set (code page 1252) Unicode values in the Unicode encoding it reports to the system;
- add Windows character set (CharSet 0) to the list of CharSets supported by the font; and
- consider the value of the encoding ID to be a Windows CharSet value and add it to the list of CharSets supported by the font.

Note that the 'cmap' subtable shall use Format 0 or 6 for its subtable, and the encoding shall be identical to the CFF's encoding.

This 'cmap' encoding is not required. It provides a compatibility mechanism for non-Unicode applications that use the font as if it were Windows ANSI encoded. Non-Windows ANSI Type 1 fonts, such as Cyrillic and Central European fonts, that Adobe shipped in the past had "0" (Windows ANSI) recorded in the CharSet field of the .PFM file; ATM for Windows 9x ignores the CharSet altogether. Adobe provides this compatibility 'cmap' encoding in every OTF converted from a Type1 font in which the Encoding is not StandardEncoding.

5.2.2.2 cmap subtable formats

5.2.2.2.1 Use of the language field in 'cmap' subtables

The language field must be set to zero for all cmap subtables whose platform IDs are other than Macintosh (platform ID 1). For cmap subtables whose platform IDs are Macintosh, set this field to the Macintosh language ID of the cmap subtable plus one, or to zero if the cmap subtable is not language-specific. For example, a Mac OS Turkish cmap subtable must set this field to 18, since the Macintosh language ID for Turkish is 17. A Mac OS Roman cmap subtable must set this field to 0, since Mac OS Roman is not a language-specific encoding.

5.2.2.2.2 Format 0: Byte encoding table

This is the Macintosh platform standard character to glyph index mapping table which is available via the Reference [7].

'cmap' Subtable Format 0:

Type	Name	Description
uint16	format	Format number is set to 0.
uint16	length	This is the length in bytes of the subtable.

uint16	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint8	glyphIdArray[256]	An array that maps character codes to glyph index values.

This is a simple 1 to 1 mapping of character codes to glyph indices. The glyph set is limited to 256. If this format is used to index into a larger glyph set, only the first 256 glyphs will be accessible.

5.2.2.2.3 Format 2: High byte mapping through table

This subtable is useful for the national character code standards used for Japanese, Chinese, and Korean characters. These code standards use a mixed 8/16-bit encoding, in which certain byte values signal the first byte of a 2-byte character (but these values are also legal as the second byte of a 2-byte character).

In addition, even for the 2-byte characters, the mapping of character codes to glyph index values depends heavily on the first byte. Consequently, the table begins with an array that maps the first byte to a 4-word subHeader. For 2-byte character codes, the subHeader is used to map the second byte's value through a subArray, as described below. When processing mixed 8/16-bit text, subHeader 0 is special: it is used for single-byte character codes. When subHeader zero is used, a second byte is not needed; the single byte value is mapped through the subArray.

'cmap' Subtable Format 2:

Type	Name	Description
uint16	format	Format number is set to 2.
uint16	length	This is the length in bytes of the subtable.
uint16	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint16	subHeaderKeys[256]	Array that maps high bytes to subHeaders: value is subHeader index * 8.
SubHeader	subHeaders[]	Variable-length array of subHeader structures.
uint16	glyphIndexArray[]	Variable-length array containing subarrays used for mapping the low byte of 2-byte characters.

A subHeader is structured as follows:

SubHeader Record:

Type	Name	Description
uint16	firstCode	First valid low byte for this subHeader.
uint16	entryCount	Number of valid low bytes for this subHeader.
int16	idDelta	See text below.
uint16	idRangeOffset	See text below.

The firstCode and entryCount values specify a subrange that begins at firstCode and has a length equal to the value of entryCount. This subrange stays within the 0-255 range of the byte being mapped. Bytes outside of

this subrange are mapped to glyph index 0 (missing glyph). The offset of the byte within this subrange is then used as index into a corresponding subarray of glyphIndexArray. This subarray is also of length entryCount. The value of the idRangeOffset is the number of bytes past the actual location of the idRangeOffset word where the glyphIndexArray element corresponding to firstCode appears.

Finally, if the value obtained from the subarray is not 0 (which indicates the missing glyph), you should add idDelta to it in order to get the glyphIndex. The value idDelta permits the same subarray to be used for several different subheaders. The idDelta arithmetic is modulo 65536.

5.2.2.2.4 Format 4: Segment mapping to delta values

This is the standard character-to-glyph-index mapping table for the Windows platform for fonts that support Unicode BMP characters. See Windows platform (platform ID = 3) in subclause 5.2.2.1 for additional details regarding subtable formats for Unicode encoding on the Windows platform.

This format is used when the character codes for the characters represented by a font fall into several contiguous ranges, possibly with holes in some or all of the ranges (that is, some of the codes in a range may not have a representation in the font). The format-dependent data is divided into three parts, which shall occur in the following order:

- a) A four-word header gives parameters for an optimized search of the segment list;
- b) Four parallel arrays describe the segments (one segment for each contiguous range of codes);
- c) A variable-length array of glyph IDs (unsigned words).

'cmap' Subtable Format 4:

Type	Name	Description
uint16	format	Format number is set to 4.
uint16	length	This is the length in bytes of the subtable.
uint16	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint16	segCountX2	2 x segCount.
uint16	searchRange	$2 \times (2^{\lceil \log_2(\text{segCount}) \rceil})$
uint16	entrySelector	$\log_2(\text{searchRange}/2)$
uint16	rangeShift	2 x segCount - searchRange
uint16	endCode[segCount]	End characterCode for each segment, last=0xFFFF.
uint16	reservedPad	Set to 0.
uint16	startCode[segCount]	Start character code for each segment.
int16	idDelta[segCount]	Delta for all character codes in segment.
uint16	idRangeOffset[segCount]	Offsets into glyphIdArray or 0
uint16	glyphIdArray[]	Glyph index array (arbitrary length)

The number of segments is specified by `segCount`, which is not explicitly in the header; however, all of the header parameters are derived from it. The `searchRange` value is twice the largest power of 2 that is less than or equal to `segCount`. For example, if `segCount=39`, we have the following:

```
segCountX2    78
searchRange   64  (= 2 x (largest power of 2 <=39))
entrySelector 5   (= log2 (32))
rangeShift    14  (= 2 x 39 – 64)
```

Each segment is described by a `startCode` and `endCode`, along with an `idDelta` and an `idRangeOffset`, which are used for mapping the character codes in the segment. The segments are sorted in order of increasing `endCode` values, and the segment values are specified in four parallel arrays. You search for the first `endCode` that is greater than or equal to the character code you want to map. If the corresponding `startCode` is less than or equal to the character code, then you use the corresponding `idDelta` and `idRangeOffset` to map the character code to a glyph index (otherwise, the `missingGlyph` is returned). For the search to terminate, the final `startCode` and `endCode` values shall be `0xFFFF`. This segment need not contain any valid mappings. (It can just map the single character code `0xFFFF` to `missingGlyph`). However, the segment must be present.

If the `idRangeOffset` value for the segment is not 0, the mapping of character codes relies on `glyphIdArray`. The character code offset from `startCode` is added to the `idRangeOffset` value. This sum is used as an offset from the current location within `idRangeOffset` itself to index out the correct `glyphIdArray` value. This obscure indexing trick works because `glyphIdArray` immediately follows `idRangeOffset` in the font file. The C expression that yields the glyph index is:

```
glyphId = *(idRangeOffset[i]/2
           + (c - startCount[i])
           + &idRangeOffset[i])
```

The value `c` is the character code in question, and `i` is the segment index in which `c` appears. If the value obtained from the indexing operation is not 0 (which indicates `missingGlyph`), `idDelta[i]` is added to it to get the glyph index. The `idDelta` arithmetic is modulo 65536.

If the `idRangeOffset` is 0, the `idDelta` value is added directly to the character code offset (i.e. `idDelta[i] + c`) to get the corresponding glyph index. Again, the `idDelta` arithmetic is modulo 65536.

As an example, the variant part of the table to map characters 10-20, 30-90, and 153-480 onto a contiguous range of glyph indices may look like this:

```
segCountX2:  8
searchRange: 8
entrySelector: 4
rangeShift:  0
endCode:     20   90   480  0xffff
reservedPad: 0
startCode:   10   30   153  0xffff
idDelta:     -9  -18  -80   1
idRangeOffset: 0   0   0   0
```

This table performs the following mappings:

```
10 -> 10 - 9 = 1
20 -> 20 - 9 = 11
30 -> 30 - 18 = 12
90 -> 90 - 18 = 72
```

Note that the delta values could be reworked so as to reorder the segments.

5.2.2.2.5 Format 6: Trimmed table mapping

'cmap' Subtable Format 6:

Type	Name	Description
uint16	format	Format number is set to 6.
uint16	length	This is the length in bytes of the subtable.
uint16	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint16	firstCode	First character code of subrange.
uint16	entryCount	Number of character codes in subrange.
uint16	glyphIdArray [entryCount]	Array of glyph index values for character codes in the range.

The firstCode and entryCount values specify a subrange (beginning at firstCode, length = entryCount) within the range of possible character codes. Codes outside of this subrange are mapped to glyph index 0. The offset of the code (from the first code) within this subrange is used as index to the glyphIdArray, which provides the glyph index value.

NOTE Supporting 4-byte character codes: While the four existing 'cmap' subtable formats which currently exist have served us well, the introduction of the Surrogates Area in the Unicode Standard has stressed them past the point of utility. This clause specifies three formats, format 8, 10 and 12; which directly support 4-byte character codes. A major change introduced with these three formats is a more pure 32-bit orientation. The 'cmap' table version number will continue to be 0x0000, for those fonts that make use of these formats.

5.2.2.2.6 Format 8: mixed 16-bit and 32-bit coverage

Format 8 is similar to format 2 in that it provides for mixed-length character codes. Instead of allowing for 8- and 16-bit character codes, however, it allows for 16- and 32-bit character codes.

If a font contains Unicode supplementary-plane characters (U+10000 to U+10FFFF), then it's likely that it will also include Unicode BMP characters (U+0000 to U+FFFF) as well. Hence, there is a need to map a mixture of 16-bit and 32-bit character codes. A simplifying assumption is made: namely, that there are no 32-bit character codes which share the same first 16 bits as any 16-bit character code. (Since the Unicode code space extends only to U+1FFFFFF, a potential conflict exists only for characters U+0000 to U+001F, which are non-printing control characters.) This means that the determination as to whether a particular 16-bit value is a standalone character code or the start of a 32-bit character code can be made by looking at the 16-bit value directly, with no further information required.

'cmap' Subtable Format 8:

Type	Name	Description
uint16	format	Subtable format; set to 8.
uint16	reserved	Reserved; set to 0
uint32	length	Byte length of this subtable (including the header)
uint32	language	For requirements on use of the language field, see subclause 5.2.2.2.1.

uint8	is32[8192]	Tightly packed array of bits (8K bytes total) indicating whether the particular 16-bit (index) value is the start of a 32-bit character code
uint32	numGroups	Number of groupings which follow
SequentialMapGroup	groups[numGroups]	Array of SequentialMapGroup records.

Each sequential map group record specifies a character range and the starting glyph ID mapped from the first character. Glyph IDs for subsequent characters follow in sequence.

SequentialMapGroup Record:

Type	Name	Description
uint32	startCharCode	First character code in this group; note that if this group is for one or more 16-bit character codes (which is determined from the is32 array), this 32-bit value will have the high 16-bits set to zero
uint32	endCharCode	Last character code in this group; same condition as listed above for the startCharCode
uint32	startGlyphID	Glyph index corresponding to the starting character code

A few notes here. The endCharCode is used, rather than a count, because comparisons for group matching are usually done on an existing character code, and having the endCharCode be there explicitly saves the necessity of an addition per group. Groups shall be sorted by increasing startCharCode. A group's endCharCode shall be less than the startCharCode of the following group, if any.

To determine if a particular word (cp) is the first half of 32-bit code points, one can use an expression such as $(\text{is32}[\text{cp} / 8] \& (1 \ll (7 - (\text{cp} \% 8))))$. If this is non-zero, then the word is the first half of a 32-bit code point.

0 is not a special value for the high word of a 32-bit code point. A font may not have both a glyph for the code point 0x0000 and glyphs for code points with a high word of 0x0000.

The presence of the packed array of bits indicating whether a particular 16-bit value is the start of a 32-bit character code is useful even when the font contains no glyphs for a particular 16-bit start value. This is because the system software often needs to know how many bytes ahead the next character begins, even if the current character maps to the missing glyph. By including this information explicitly in this table, no "secret" knowledge needs to be encoded into the OS.

Although this format was created to support Unicode supplementary-plane characters, it's not widely supported or used. Also, no character encoding other than Unicode uses mixed 16-/32-bit characters. The use of this format is discouraged.

5.2.2.2.7 Format 10: Trimmed array

Format 10 is similar to format 6, in that it defines a trimmed array for a tight range of character codes. It differs, however, in that it uses 32-bit character codes.

'cmap' Subtable Format 10:

Type	Name	Description
uint16	format	Subtable format; set to 10.
uint16	reserved	Reserved; set to 0
uint32	length	Byte length of this subtable (including the header)
uint32	language	For requirements on use of the language field, see subclause

		5.2.2.2.1.
uint32	startCharCode	First character code covered
uint32	numChars	Number of character codes covered
uint16	glyphs[]	Array of glyph indices for the character codes covered

This format is not widely used and is not supported on Windows platforms. It would be most suitable for fonts that support only a contiguous range of Unicode supplementary-plane characters, but such fonts are rare.

5.2.2.2.8 Format 12: Segmented coverage

This is the standard character-to-glyph-index mapping table for the Windows platform for fonts supporting Unicode supplementary-plane characters (U+10000 to U+10FFFF). See Windows platform (platform ID = 3) in subclause 5.2.2.1 for additional details regarding subtable formats for Unicode encoding on the Windows platform.

Format 12 is similar to format 4 in that it defines segments for sparse representation in 4-byte character space. It differs, however, in that it uses 32-bit character codes.

'cmap' Subtable Format 12:

Type	Name	Description
uint16	format	Subtable format; set to 12.
uint16	reserved	Reserved; set to 0
uint32	length	Byte length of this subtable (including the header)
uint32	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint32	numGroups	Number of groupings which follow
SequentialMapGroup	groups[numGroups]	Array of SequentialMapGroup records.

The sequential map group record is the same format as is used for the format 8 subtable. The qualifications regarding 16-bit character codes does not apply here, however, since characters codes are uniformly 32-bit.

SequentialMapGroup Record:

Type	Name	Description
uint32	startCharCode	First character code in this group
uint32	endCharCode	Last character code in this group
uint32	startGlyphID	Glyph index corresponding to the starting character code

Groups shall be sorted by increasing startCharCode. A group's endCharCode shall be less than the startCharCode of the following group, if any. The endCharCode is used, rather than a count, because comparisons for group matching are usually done on an existing character code, and having the endCharCode be there explicitly saves the necessity of an addition per group.

5.2.2.2.9 Format 13: Many-to-one range mappings

This subtable provides for situations in which the same glyph is used for hundreds or even thousands of consecutive characters spanning across multiple ranges of the code space. This subtable format may be useful for “last resort” fonts, although these fonts may use other suitable subtable formats as well. (For “last resort” fonts, see also the ‘head’ table flags in [subclause 5.2.3](#), flags: bit 14).

NOTE Subtable formats 13 has the same data structure as format 12; it differs only in the interpretation of the startGlyphID/glyphID fields.

‘cmap’ Subtable Format 13:

Type	Name	Description
uint16	format	Subtable format; set to 13.
uint16	reserved	Reserved; set to 0
uint32	length	Byte length of this subtable (including the header)
uint32	language	For requirements on use of the language field, see subclause 5.2.2.2.1.
uint32	numGroups	Number of groupings which follow
ConstantMapGroup	groups[numGroups]	Array of ConstantMapGroup records.

The constant map group record has the same structure as the sequential map group record, with start and end character codes and a mapped glyph ID. However, the same glyph ID applies to all characters in the specified range rather than sequential glyph IDs.

ConstantMapGroup Record:

Type	Name	Description
uint32	startCharCode	First character code in this group
uint32	endCharCode	Last character code in this group
uint32	glyphID	Glyph index to be used for all the characters in the group's range.

5.2.2.2.10 Format 14: Unicode variation sequences

Subtable format 14 specifies the Unicode Variation Sequences (UVSes) supported by the font. A Variation Sequence, according to the Unicode Standard, comprises a base character followed by a variation selector; e.g. <U+82A6, U+E0101>.

The subtable partitions the UVSes supported by the font into two categories: “default” and “non-default” UVSes. Given a UVS, if the glyph obtained by looking up the base character of that sequence in the Unicode cmap subtable (i.e. the BMP subtable or BMP + supplementary-planes subtable) is the glyph to use for that sequence, then the sequence is a “default” UVS. Otherwise, it is a “non-default” UVS, and the glyph to use for that sequence is specified in the format 14 subtable itself.

The example in the end of this subclause shows how a font vendor can use format 14 for a JIS-2004-aware font.

'cmap' Subtable Format 14: Format 14 header		
Type	Name	Description
uint16	format	Subtable format; set to 14.
uint32	length	Byte length of this subtable (including this header)
uint32	numVarSelectorRecords	Number of variation Selector Records
VariationSelector	varSelector[numVarSelectorRecords]	Array of VariationSelector records.

Each variation selector records specifies a variation selector character, and offsets to “default” and “non-default” tables used to map variation sequences using that variation selector.

Variation Selector Record		
Type	Name	Description
uint24	varSelector	Variation selector
Offset32	defaultUVSOffset	Offset from the start of the format 14 subtable to Default UVS Table. May be 0.
Offset32	nonDefaultUVSOffset	Offset from the start of the format 14 subtable to Non-Default UVS Table. May be 0.

The Variation Selector Records are sorted in increasing order of ‘varSelector’. No two records may have the same ‘varSelector’.

A Variation Selector Record and the data its offsets point to specify those UVSes supported by the font for which the variation selector is the ‘varSelector’ value of the record. The base characters of the UVSes are stored in the tables pointed to by the offsets. The UVSes are partitioned by whether they are default or non-default UVSes.

Glyph IDs to be used for non-default UVSes are specified in the Non-Default UVS table.

Default UVS table

A Default UVS Table is simply a range-compressed list of Unicode scalar values, representing the base characters of the default UVSes which use the ‘varSelector’ of the associated Variation Selector Record.

Default UVS Table header		
Type	Name	Description
uint32	numUnicodeValueRanges	Number of ranges that follow
UnicodeRange	ranges[numUnicodeValueRanges]	Array of UnicodeRange records.

Each Unicode range record specifies a contiguous range of Unicode values.

UnicodeRange Record		
Type	Name	Description
uint24	startUnicodeValue	First value in this range
uint8	additionalCount	Number of additional values in this range

For example, the range U+4E4D–U+4E4F (3 values) will set ‘startUnicodeValue’ to 0x004E4D and ‘additionalCount’ to 2. A singleton range will set ‘additionalCount’ to 0.

(‘startUnicodeValue’ + ‘additionalCount’) shall not exceed 0xFFFFFFFF.

The Unicode Value Ranges are sorted in increasing order of 'startUnicodeValue'. The ranges must not overlap; i.e., ('startUnicodeValue' + 'additionalCount') must be less than the 'startUnicodeValue' of the following range (if any).

Non-default UVS table

A Non-Default UVS Table is a list of pairs of Unicode scalar values and glyph IDs. The Unicode values represent the base characters of all non-default UVSes which use the 'varSelector' of the associated Variation Selector Record, and the glyph IDs specify the glyph IDs to use for the UVSes.

Non-Default UVS Table header		
Type	Name	Description
uint32	numUVSMappings	Number of UVS Mappings that follow
UVSMapping	uvsMappings[numUVSMappings]	Array of UVSMapping records.

Each UVSMapping record provides a glyph ID mapping for one base Unicode character, when that base character is used in a variation sequence with the current variation selector.

UVSMapping Record		
Type	Name	Description
uint24	unicodeValue	Base Unicode value of the UVS
uint16	glyphID	Glyph ID of the UVS

The UVS Mappings are sorted in increasing order of 'unicodeValue'. No two mappings in this table may have the same 'unicodeValue' values.

Example

Here is an example of how a format 14 cmap subtable may be used in a font that is aware of JIS-2004 variant glyphs. The CIDs (character IDs) in this example refer to those in the Adobe Character Collection "Adobe-Japan1", and may be assumed to be identical to the glyph IDs in the font in our example.

JIS-2004 changed the default glyph variants for some of its code points. For example:

JIS-90: U+82A6 -> CID 1142

JIS-2004: U+82A6 -> CID 7961

Both of these glyph variants are supported through the use of Unicode Variation Sequences, as the following examples from Unicode's UVS registry show:

U+82A6 U+E0100 -> CID 1142

U+82A6 U+E0101 -> CID 7961

If the font wants to support the JIS-2004 variants by default, it will:

- encode glyph ID 7961 at U+82A6 in the Unicode cmap subtable,
- specify in the UVS cmap subtable's Default UVS Table ('varSelector' will be 0x0E0101 and 'defaultUVSOffset' will point to data containing a 0x0082A6 Unicode value)
- specify -> glyph ID 1142 in the UVS cmap subtable's Non-Default UVS Table ('varSelector' will be 0x0E0100 and 'nonDefaultBaseUVOffset' will point to data containing a 'unicodeValue' 0x82A6 and 'glyphID' 1142).

If, however, the font wants to support the JIS-90 variants by default, it will:

- encode glyph ID 1142 at U+82A6 in the Unicode cmap subtable,
- specify in the UVS cmap subtable's Default UVS Table
- specify -> glyph ID 7961 in the UVS cmap subtable's Non-Default UVS Table

5.2.3 head – Font header

This table gives global information about the font. The bounding box values should be computed using *only* glyphs that have contours. Glyphs with no contours should be ignored for the purposes of these calculations.

Type	Name	Description
uint16	majorVersion	Major version number of the font header table – set to 1.
uint16	minorVersion	Minor version number of the font header table – set to 0.
Fixed	fontRevision	Set by font manufacturer.
uint32	checksumAdjustment	To compute: set it to 0, sum the entire font as uint32, then store 0xB1B0AFBA - sum. If the font is used as a component in a font collection file, the value of this field will be invalidated by changes to the file structure and font table directory, and must be ignored.
uint32	magicNumber	Set to 0x5F0F3CF5.
uint16	flags	Bit 0: Baseline for font at y=0; Bit 1: Left sidebearing point at x=0 (relevant only for TrueType rasterizers) – see the note below regarding variable fonts; Bit 2: Instructions may depend on point size; Bit 3: Force ppeM to integer values for all internal scaler math; may use fractional ppeM sizes if this bit is clear; Bit 4: Instructions may alter advance width (the advance widths might not scale linearly); Bit 5: This bit is not used in OFF, and should not be set in order to ensure compatible behavior on all platforms. If set, it may result in different behavior for vertical layout in some platforms. (See 'head' table specification of "Apple's TrueType Reference Manual" [7] for details regarding behavior in Apple platforms.) Bits 6-10: These bits are not used in OFF and should always be cleared. (See 'head' table specification of "Apple's TrueType Reference Manual" [7] for details regarding legacy use in Apple platforms.) Bit 11: Font data is 'lossless' as a result of having been subjected to optimizing transformation and/or compression (such as font compression mechanisms defined by ISO/IEC 14496-18, MicroType⁵ Express, WOFF2 [24], or similar) where the original font functionality and features are retained but the binary compatibility between input and output font files is not guaranteed. As a result of the applied transform, the 'DSIG' Table may also be invalidated. Bit 12: Font converted (produce compatible metrics) Bit 13: Font optimized for ClearType⁶. Note, fonts that rely on embedded bitmaps (EBDT) for rendering should not be considered optimized for ClearType, and therefore

⁵ MicroType® is the trademark of a product supplied by Monotype Imaging Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of this product.

⁶ ClearType® is the trademark of a product supplied by Microsoft. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of this product.

		should keep this bit cleared. Bit 14: Last Resort font. If set, indicates that the glyphs encoded in the cmap subtables are simply generic symbolic representations of code point ranges and don't truly represent support for those code points. If unset, indicates that the glyphs encoded in the cmap subtables represent proper support for those code points Bit 15: Reserved, set to 0
uint16	unitsPerEm	Set to a value from 16 to 16384. Any value in this range is valid. In fonts that have TrueType outlines, a power of 2 is recommended as this allows performance optimization in some rasterizers.
LONGDATETIME	created	Number of seconds since 12:00 midnight that started January 1 st , 1904, in GMT/UTC time zone. 64-bit integer
LONGDATETIME	modified	Number of seconds since 12:00 midnight that started January 1 st , 1904, in GMT/UTC time zone. 64-bit integer
int16	xMin	For all glyph bounding boxes.
int16	yMin	For all glyph bounding boxes.
int16	xMax	For all glyph bounding boxes.
int16	yMax	For all glyph bounding boxes.
uint16	macStyle	Bit 0: Bold (if set to 1); Bit 1: Italic (if set to 1) Bit 2: Underline (if set to 1) Bit 3: Outline (if set to 1) Bit 4: Shadow (if set to 1) Bit 5: Condensed (if set to 1) Bit 6: Extended (if set to 1) Bits 7-15: Reserved (set to 0).
uint16	lowestRecPPEM	Smallest readable size in pixels.
int16	fontDirectionHint	Deprecated (Set to 2). 0: Fully mixed directional glyphs; 1: Only strongly left to right; 2: Like 1 but also contains neutrals; -1: Only strongly right to left; -2: Like -1 but also contains neutrals. ¹
int16	indexToLocFormat	0 for short offsets, 1 for long.
int16	glyphDataFormat	0 for current format.

A neutral character has no inherent directionality; it is not a character with zero (0) width. Spaces and punctuation are examples of neutral characters. Non-neutral characters are those with inherent directionality. For example, Roman letters (left-to-right) and Arabic letters (right-to-left) have directionality. In a "normal" Roman font where spaces and punctuation are present, the font direction hints should be set to two (2).

It should be noted that the macStyle bits must agree with the 'OS/2' table fsSelection bits. The fsSelection bits are used over the macStyle bits in Windows. The PANOSE values and 'post' table values are ignored for determining bold or italic fonts.

For historical reasons, the fontRevision value contained in this table is not used by Windows to determine the version of a font. Instead, Windows evaluates the version string (id 5) in the 'name' table.

Note that, in a variable font with TrueType outlines, the left side bearing for each glyph must equal xMin, and bit 1 in the flags field must be set. Also, bit 5 must be cleared in all variable fonts. For general information on OFF Font variations, see [subclause 7.1](#).

5.2.4 hhea – Horizontal header

This table contains information for horizontal layout. The values in the minRightSidebearing, minLeftSidebearing and xMaxExtent should be computed using *only* glyphs that have contours. Glyphs with no contours should be ignored for the purposes of these calculations. All reserved areas shall be set to 0.

Type	Name	Description
uint16	majorVersion	Major version number of the horizontal header table – set to 1.
uint16	minorVersion	Minor version number of the horizontal header table – set to 0.
FWORD	ascender	Typographic ascent. (Distance from baseline of highest ascender)
FWORD	descender	Typographic descent. (Distance from baseline of lowest descender)
FWORD	lineGap	Typographic line gap. Negative lineGap values are treated as zero in some legacy platform implementations.
UWORD	advanceWidthMax	Maximum advance width value in 'hmtx' table.
FWORD	minLeftSidebearing	Minimum left sidebearing value in 'hmtx' table.
FWORD	minRightSidebearing	Minimum right sidebearing value; calculated as $\text{Min}(\text{aw} - \text{lsb} - (\text{xMax} - \text{xMin}))$.
FWORD	xMaxExtent	$\text{Max}(\text{lsb} + (\text{xMax} - \text{xMin}))$.
int16	caretSlopeRise	Used to calculate the slope of the cursor (rise/run); 1 for vertical.
int16	caretSlopeRun	0 for vertical.
int16	caretOffset	The amount by which a slanted highlight on a glyph needs to be shifted to produce the best appearance. Set to 0 for non-slanted fonts
int16	(reserved)	set to 0
int16	(reserved)	set to 0
int16	(reserved)	set to 0
int16	(reserved)	set to 0

int16	metricDataFormat	0 for current format.
uint16	numberOfHMetrics	Number of hMetric entries in 'hmtx' table

NOTE The ascender, descender and linegap values in this table are Macintosh platform specific. These are not ignored by Windows platform. They are used to identify fixed pitch fonts. Also see information in the OS/2 table.

'hhea' Table and OFF Font Variations

In a variable font, various font-metric values within the horizontal header table may need to be adjusted for different variation instances. Variation data for 'hhea' entries can be provided in the [metrics variations \('MVAR'\) table](#). Different 'hhea' entries are associated with particular variation data in the 'MVAR' table using value tags, as follows:

'hhea' entry	Tag
caretOffset	'hcof'
caretSlopeRise	'hcrs'
caretSlopeRun	'hcrn'

For general information on OFF Font variations, see [subclause 7.1](#).

5.2.5 hmtx – Horizontal metrics

Glyph metrics used for horizontal text layout include glyph advance widths, side bearings and X-direction min and max values (xMin, xMax). These are derived using a combination of the glyph outline data ('glyf', 'CFF' or 'CFF2') and the horizontal metrics table. The horizontal metrics ('hmtx') table provides glyph advance widths and left side bearings.

In a font with TrueType outline data, the ['glyf'](#) table provides xMin and xMax values, but not advance widths or side bearings. The advance width is always obtained from the 'hmtx' table. In some fonts, depending on the state of flags in the ['head'](#) table, the left side bearings may be the same as the xMin values in the 'glyf' table, though this is not true for all fonts. (See the description of bit 1 of the flags field in the 'head' table.) For this reason, left side bearings are provided in the 'hmtx' table. The right side bearing is always derived using advance width and left side bearing values from the 'hmtx' table, plus bounding-box information in the glyph description — see below for more details.

In a variable font with TrueType outline data, the left side bearing value in the 'hmtx' table must always be equal to xMin (bit 1 of the 'head' flags field must be set). Hence, these values can also be derived directly from the 'glyf' table. Note that these values apply only to the default instance of the variable font: non-default instances may have different side bearing values. These can be derived from interpolated “phantom point” coordinates using the ['gvar'](#) table (see below for additional details), or by applying variation data in the ['HVAR'](#) table to default-instance values from the 'glyf' or 'hmtx' table.

In a font with CFF version 1 outline data, the 'CFF' table does include advance widths. These values are used by PostScript processors, but are not used in OFF layout. In an OFF context, the 'hmtx' table is required and must be used for advance widths. Note that fonts in a Font Collection file that share a 'CFF' table may specify different advance widths in font-specific 'hmtx' tables for a particular glyph index. Also note that the 'CFF2' table does not include advance widths. In addition, for either 'CFF' or 'CFF2' data, there are no explicit xMin and xMax values; side bearings are implicitly contained within the CharString data, and can be obtained from the the CFF / CFF2 rasterizer. Some layout engines may use left side bearing values in the 'hmtx' table, however; hence, font production tools should ensure that the lsb values in the 'hmtx' table match the implicit xMin values reflected in the CharString data. In a variable font with CFF2 outline data, left side bearing and advance width values for non-default instances should be obtained by combining information from the 'hmtx' and 'HVAR' tables.

The table uses a longHorMetric record to give the advance width and left side bearing of a glyph. Records are indexed by glyph ID. As an optimization, the number of records can be less than the number of glyphs, in which case the advance width value of the last record applies to all remaining glyph IDs. This can be useful in monospaced fonts, or in fonts that have a large number of glyphs with the same advance width (provided the glyphs are ordered appropriately). The number of longHorMetric records is determined by the numberOfHMetrics field in the ['hhea'](#) table.

If the longHorMetric array is less than the total number of glyphs, then that array is followed by an array for the left side bearing values of the remaining glyphs. The number of elements in the left side bearing will be derived from numberOfHMetrics plus the numGlyphs field in the ['maxp'](#) table.

Horizontal Metrics Table:

Type	Name	Description
longHorMetric	hMetrics [numberOfHMetrics]	Paired advance width and left side bearing values for each glyph. Records are indexed by glyph ID.
int16	leftSideBearing [numGlyphs - numberOfHMetrics]	Left side bearings for glyph IDs greater than or equal to numberOfHMetrics.

longHorMetric Record:

Type	Name	Description
uint16	advanceWidth	Advance width, in font design units.
int16	lsb	Glyph left side bearing, in font design units.

In a font with TrueType outlines, xMin and xMax values for each glyph are given in the 'glyf' table. The advance width ("aw") and left side bearing ("lsb") can be derived from the glyph "phantom points", which are computed by the TrueType rasterizer; or they can be obtained from the 'hmtx' table. In a font with CFF or CFF2 outlines, xMin (= left side bearing) and xMax values can be obtained from the CFF / CFF2 rasterizer. From those values, the right side bearing ("rsb") is calculated as follows:

$$rsb = aw - (lsb + xMax - xMin)$$

If pp1 and pp2 are TrueType phantom points used to control lsb and rsb, their initial position in the X-direction is calculated as follows:

$$\begin{aligned} pp1 &= xMin - lsb \\ pp2 &= pp1 + aw \end{aligned}$$

5.2.6 maxp – Maximum profile

This table establishes the memory requirements for this font. Fonts with CFF data must use Version 0.5 of this table, specifying only the numGlyphs field. Fonts with TrueType outlines must use Version 1.0 of this table, where all data is required.

Version 0.5

Type	Name	Description
Fixed	version	0x00005000 for version 0.5 (Note the difference in the representation of a non-zero fractional part, in Fixed numbers.)
uint16	numGlyphs	The number of glyphs in the font.

Version 1.0

Type	Name	Description
Fixed	version	0x00010000 for version 1.0.
uint16	numGlyphs	The number of glyphs in the font.
uint16	maxPoints	Maximum points in a non-composite glyph.
uint16	maxContours	Maximum contours in a non-composite glyph.
uint16	maxCompositePoints	Maximum points in a composite glyph.
uint16	maxCompositeContours	Maximum contours in a composite glyph.
uint16	maxZones	1 if instructions do not use the twilight zone (Z0), or 2 if instructions do use Z0; should be set to 2 in most cases.
uint16	maxTwilightPoints	Maximum points used in Z0.
uint16	maxStorage	Number of Storage Area locations.
uint16	maxFunctionDefs	Number of FDEFs, equal to the highest function number + 1.
uint16	maxInstructionDefs	Number of IDEFs.
uint16	maxStackElements	Maximum stack depth across Font Program ('fpgm' table), CVT Program ('prep' table) and all glyph instructions (in the 'glyf' table).
uint16	maxSizeOfInstructions	Maximum byte count for glyph instructions.
uint16	maxComponentElements	Maximum number of components referenced at "top level" for any composite glyph.
uint16	maxComponentDepth	Maximum levels of recursion; 1 for simple components.

5.2.7 name – Naming table

5.2.7.1 Table structure

The naming table allows multilingual strings to be associated with the OFF font file. These strings can represent copyright notices, font names, family names, style names, and so on. To keep this table short, the font manufacturer may wish to make a limited set of entries in some small set of languages; later, the font can be "localized" and the strings translated or added. Other parts of the OFF font file that require these strings can refer to them using a language-independent name ID. In addition to language variants, the table also allows for platform-specific character-encoding variants. Clients that need a particular string can look it up by its platform ID, encoding ID, language ID and name ID. Note that different platforms may have different requirements for the encoding of strings.

Many newer platforms can use strings intended for different platforms if a font does not include strings for that platform. Some applications might display incorrect strings, however, if strings for the current platform are not included.

Naming table header

There are two formats for the Naming Table. Format 0 uses platform-specific numeric language identifiers. Format 1 allows for use of language-tag strings to indicate the language of Naming-Table strings. Both formats include variable-size string-data storage, and an array of name records that are used to identify the type of string (name ID), platform, encoding and language variants of the string, and the location within the storage.

Naming table format 0

The format 0 naming table is organized as follows:

Type	Name	Description
uint16	format	Format selector (=0).
uint16	count	Number of name records.
Offset16	stringOffset	Offset to start of string storage (from start of table).
NameRecord	nameRecord[count]	The name records where <i>count</i> is the number of records.
(Variable)		Storage for the actual string data.

Format 0 differs from format 1 in regard to handling of language identification: it uses only numeric language IDs, which generally are values less than 0x8000 and have platform-specific interpretations. See [subclause 5.2.7.2](#) for more details.

Naming table format 1

The format 1 naming table adds additional elements, as follows:

Type	Name	Description
uint16	format	Format selector (=1).
uint16	count	Number of name records.
Offset16	stringOffset	Offset to start of string storage (from start of table).
NameRecord	nameRecord[count]	The name records where <i>count</i> is the number of records.
uint16	langTagCount	Number of language-tag records.
LangTagRecord	langTagRecord[langTagCount]	The language-tag records where <i>langTagCount</i> is the number of records.
(Variable)		Storage for the actual string data.

When format 1 is used, the language IDs in name records can be less than or greater than 0x8000. If a language ID is less than 0x8000, it has a platform-specific interpretation as with a format 0 naming table. If a language ID is equal to or greater than 0x8000, it is associated with a language-tag record (LangTagRecord) that references a language-tag string. In this way, the language ID is associated with a language-tag string that specifies the language for name records using that language ID, regardless of the platform. These can be used for any platform that supports this language-tag mechanism.

A font using a format 1 naming table may use a combination of platform-specific language IDs as well as language-tag records for a given platform and encoding.

Each *LangTagRecord* is organized as follows:

Type	Name	Description
uint16	length	Language-tag string length (in bytes).
uint16	offset	Language-tag string offset from start of storage area (in bytes).

Language-tag strings stored in the naming table must be encoded in UTF-16BE. The language tags shall be as specified in the IETF specification BCP 47. This provides tags such as "en", "fr-CA" and "zh-Hant" to identify languages, including dialects, written form and other language variants.

The language-tag records are associated sequentially with language IDs starting with 0x8000. Each language-tag record corresponds to a language ID one greater than that for the previous language-tag record. Thus, language IDs associated with language-tag records must be within the range 0x8000 to 0x8000 + langTagCount - 1. If a name record uses a language ID that is greater than this, the identity of the language is unknown; such name records should not be used.

For example, suppose a font has two language-tag records referencing strings in the storage: the first references the string "en", and the second references the string "zh-Hant-HK". In this case, the language ID 0x8000 is used in name records to index English-language strings. The language ID 0x8001 is used in name records to index strings in Traditional Chinese as used in Hong Kong.

5.2.7.2 Name records

Each string in the string storage is referenced by a name record. The name record has a multi-part key, to identify the logical type of string and its language or platform-specific implementation variants, plus the location of the string in the string storage.

Each *NameRecord* is organized as follows:

Type	Name	Description
uint16	platformID	Platform ID.
uint16	encodingID	Platform-specific encoding ID.
uint16	languageID	Language ID.
uint16	nameID	Name ID.
uint16	length	String length (in bytes).
Offset16	Offset	String offset from start of storage area (in bytes).

The name ID identifies a logical string category, such as family name or copyright. Name IDs are the same for all platforms and languages; these are described in detail below. The other three elements of the key allow for platform-specific implementations: a platform ID, a platform-specific encoding ID, and a language ID.

As with encoding records in the 'cmap' table, name records shall be sorted first by platform ID, then by encoding ID, then language ID, and then, in addition, by name ID. Descriptions of the various IDs follow.

5.2.7.3 Platform, encoding and language IDs

The platform, encoding and language IDs of a name record allow for platform-specific implementations. Different platforms can support different encodings, and different languages. All encoding IDs are platform-

specific. Language IDs are similarly platform-specific, except in the case of IDs used in conjunction with the language-tag mechanism of naming table format 1, described above.

NOTE Platform IDs, platform-specific encoding IDs and, in some cases, platform-specific language IDs are also used in the 'cmap' table (see subclause 5.2.2)

Language IDs refer to a value that identifies the language in which a particular string is written. Values less than 0x8000 are defined on a platform-specific basis. A format 0 naming table shall use only language ID values less than 0x8000 from the platform-specific enumerations given below. (Exceptions to this requirement are permitted, however, in the case of user-defined platforms — platform IDs 240 to 255.) Values greater than or equal to 0x8000 can be used in a format 1 naming table in conjunction with language-tag records, as described above. Not all platforms have platform-specific language IDs, and not all platforms support language-tag records.

Platform IDs

The following platform IDs are defined:

Platform ID	Platform name	Platform-specific encoding IDs	Language IDs
0	Unicode	Various	None
1	Macintosh	Script manager code	Various
2	<i>ISO [deprecated]</i>	<i>ISO encoding [deprecated]</i>	<i>None</i>
3	Windows	Windows encoding	Various
4	Custom	Custom	None

Platform ID 2 (ISO) has been deprecated. It was intended to represent ISO/IEC 10646, as opposed to Unicode. It is redundant, however, since both standards have identical character code assignments.

Platform ID values 240 through 255 are reserved for user-defined platforms, and shall never be assigned to a registered platform.

Platform-specific encoding and language IDs: Unicode platform (platform ID = 0)

The following encoding IDs are defined for use with the Unicode platform:

Encoding ID	Description
0	Unicode 1.0 semantics
1	Unicode 1.1 semantics
2	ISO/IEC 10646 semantics
3	Unicode 2.0 and onwards semantics, Unicode BMP only (cmap subtable formats 0, 4, 6).
4	Unicode 2.0 and onwards semantics, Unicode full repertoire (cmap subtable formats 0, 4, 6, 10, 12).
5	Unicode Variation Sequences (cmap subtable format 14).
6	Unicode full repertoire (cmap subtable formats 0, 4, 6, 10, 12, 13).

A new encoding ID for the Unicode platform will be assigned if a new version of Unicode moves characters, in order to properly specify character code semantics. (Because of Unicode stability policies, such a need is not anticipated.) The distinction between Unicode platform-specific encoding IDs 1 and 2 is for historical reasons only; The Unicode Standard is in fact identical in repertoire and encoding to ISO/IEC 10646. For all practical

purposes in current fonts, the distinctions provided by encoding IDs 0, 1 and 2 are not important, thus these encoding IDs are deprecated.

A new encoding ID for the Unicode platform is also sometimes assigned when new cmap subtable formats are added to the specification, so as to allow for compatibility with existing parsers. For example, when cmap subtable formats 10 and 12 were added to the specification, encoding ID 4 was added as well, and when cmap subtable format 13 was added to the specification, encoding ID 6 was added. The cmap subtable formats listed in the table above are the only ones that may be used for the corresponding encoding ID.

Unicode platform encoding ID 5 can be used for encodings in the 'cmap' table but not for strings in the 'name' table.

There are no platform-specific language IDs defined for the Unicode platform. Language ID = 0 may be used for Unicode-platform strings, but this does not indicate any particular language. Language IDs greater than or equal to 0x8000 may be used together with language-tag records, as described above.

Platform-specific encoding and language IDs: Windows platform (platform ID= 3)

Windows encoding IDs

The following encoding IDs are defined for use with the Windows platform:

Platform ID	Encoding ID	Description
3	0	Symbol
3	1	Unicode BMP
3	2	ShiftJIS
3	3	PRC
3	4	Big5
3	5	Wansung
3	6	Johab
3	7	Reserved
3	8	Reserved
3	9	Reserved
3	10	Unicode full repertoire

When building a Unicode font for Windows, the platform ID should be 3 and the encoding ID should be 1, and the referenced string data must be encoded in UTF-16BE. When building a symbol font for Windows, the platform ID should be 3 and the encoding ID should be 0, and the referenced string data must be encoded in UTF-16BE.

Windows language IDs

The following are platform-specific language IDs assigned by Microsoft:

Primary Language	Region	Language ID (hexadecimal)
Afrikaans	South Africa	0436
Albanian	Albania	041C
Alsatian	France	0484
Amharic	Ethiopia	045E
Arabic	Algeria	1401
Arabic	Bahrain	3C01
Arabic	Egypt	0C01
Arabic	Iraq	0801
Arabic	Jordan	2C01
Arabic	Kuwait	3401
Arabic	Lebanon	3001
Arabic	Libya	1001
Arabic	Morocco	1801
Arabic	Oman	2001
Arabic	Qatar	4001
Arabic	Saudi Arabia	0401
Arabic	Syria	2801
Arabic	Tunisia	1C01
Arabic	U.A.E.	3801
Arabic	Yemen	2401
Armenian	Armenia	042B>
Assamese	India	044D
Azeri (Cyrillic)	Azerbaijan	082C
Azeri (Latin)	Azerbaijan	042C
Bashkir	Russia	046D
Basque	Basque	042D
Belarusian	Belarus	0423
Bengali	Bangladesh	0845
Bengali	India	0445
Bosnian (Cyrillic)	Bosnia and Herzegovina	201A
Bosnian (Latin)	Bosnia and Herzegovina	141A
Breton	France	047E
Bulgarian	Bulgaria	0402
Catalan	Catalan	0403
Chinese	Hong Kong S.A.R.	0C04
Chinese	Macao S.A.R.	1404

Chinese	People's Republic of China	0804
Chinese	Singapore	1004
Chinese	Taiwan	0404
Corsican	France	0483
Croatian	Croatia	041A
Croatian (Latin)	Bosnia and Herzegovina	101A
Czech	Czech Republic	0405
Danish	Denmark	0406
Dari	Afghanistan	048C
Divehi	Maldives	0465
Dutch	Belgium	0813
Dutch	Netherlands	0413
English	Australia	0C09
English	Belize	2809
English	Canada	1009
English	Caribbean	2409
English	India	4009
English	Ireland	1809
English	Jamaica	2009
English	Malaysia	4409
English	New Zealand	1409
English	Republic of the Philippines	3409
English	Singapore	4809
English	South Africa	1C09
English	Trinidad and Tobago	2C09
English	United Kingdom	0809
English	United States	0409
English	Zimbabwe	3009
Estonian	Estonia	0425
Faroese	Faroe Islands	0438
Filipino	Philippines	0464
Finnish	Finland	040B
French	Belgium	080C
French	Canada	0C0C
French	France	040C
French	Luxembourg	140c
French	Principality of Monaco	180C
French	Switzerland	100C

Frisian	Netherlands	0462
Galician	Galician	0456
Georgian	Georgia	0437
German	Austria	0C07
German	Germany	0407
German	Liechtenstein	1407
German	Luxembourg	1007
German	Switzerland	0807
Greek	Greece	0408
Greenlandic	Greenland	046F
Gujarati	India	0447
Hausa (Latin)	Nigeria	0468
Hebrew	Israel	040D
Hindi	India	0439
Hungarian	Hungary	040E
Icelandic	Iceland	040F
Igbo	Nigeria	0470
Indonesian	Indonesia	0421
Inuktitut	Canada	045D
Inuktitut (Latin)	Canada	085D
Irish	Ireland	083C
isiXhosa	South Africa	0434
isiZulu	South Africa	0435
Italian	Italy	0410
Italian	Switzerland	0810
Japanese	Japan	0411
Kannada	India	044B
Kazakh	Kazakhstan	043F
Khmer	Cambodia	0453
K'iche	Guatemala	0486
Kinyarwanda	Rwanda	0487
Kiswahili	Kenya	0441
Konkani	India	0457
Korean	Korea	0412
Kyrgyz	Kyrgyzstan	0440
Lao	Lao P.D.R.	0454
Latvian	Latvia	0426
Lithuanian	Lithuania	0427

Lower Sorbian	Germany	082E
Luxembourgish	Luxembourg	046E
Macedonian (FYROM)	Former Yugoslav Republic of Macedonia	042F
Malay	Brunei Darussalam	083E
Malay	Malaysia	043E
Malayalam	India	044C
Maltese	Malta	043A
Maori	New Zealand	0481
Mapudungun	Chile	047A
Marathi	India	044E
Mohawk	Mohawk	047C
Mongolian (Cyrillic)	Mongolia	0450
Mongolian (Traditional)	People's Republic of China	0850
Nepali	Nepal	0461
Norwegian (Bokmal)	Norway	0414
Norwegian (Nynorsk)	Norway	0814
Occitan	France	0482
Odia (formerly Oriya)	India	0448
Pashto	Afghanistan	0463
Polish	Poland	0415
Portuguese	Brazil	0416
Portuguese	Portugal	0816
Punjabi	India	0446
Quechua	Bolivia	046B
Quechua	Ecuador	086B
Quechua	Peru	0C6B
Romanian	Romania	0418
Romansh	Switzerland	0417
Russian	Russia	0419
Sami (Inari)	Finland	243B
Sami (Lule)	Norway	103B
Sami (Lule)	Sweden	143B
Sami (Northern)	Finland	0C3B
Sami (Northern)	Norway	043B
Sami (Northern)	Sweden	083B
Sami (Skolt)	Finland	203B
Sami (Southern)	Norway	183B
Sami (Southern)	Sweden	1C3B

Sanskrit	India	044F
Serbian (Cyrillic)	Bosnia and Herzegovina	1C1A
Serbian (Cyrillic)	Serbia	0C1A
Serbian (Latin)	Bosnia and Herzegovina	181A
Serbian (Latin)	Serbia	081A
Sesotho sa Leboa	South Africa	046C
Setswana	South Africa	0432
Sinhala	Sri Lanka	045B
Slovak	Slovakia	041B
Slovenian	Slovenia	0424
Spanish	Argentina	2C0A
Spanish	Bolivia	400A
Spanish	Chile	340A
Spanish	Colombia	240A
Spanish	Costa Rica	140A
Spanish	Dominican Republic	1C0A
Spanish	Ecuador	300A
Spanish	El Salvador	440A
Spanish	Guatemala	100A
Spanish	Honduras	480A
Spanish	Mexico	080A
Spanish	Nicaragua	4C0A
Spanish	Panama	180A
Spanish	Paraguay	3C0A
Spanish	Peru	280A
Spanish	Puerto Rico	500A
Spanish (Modern sort)	Spain	0C0A
Spanish (Traditional sort)	Spain	040A
Spanish	United States	540A
Spanish	Uruguay	380A
Spanish	Venezuela	200A
Sweden	Finland	081D
Swedish	Sweden	041D
Syriac	Syria	045A
Tajik (Cyrillic)	Tajikistan	0428
Tamazight (Latin)	Algeria	085F
Tamil	India	0449
Tatar	Russia	0444

Telugu	India	044A
Thai	Thailand	041E
Tibetan	PRC	0451
Turkish	Turkey	041F
Turkmen	Turkmenistan	0442
Uighur	PRC	0480
Ukrainian	Ukraine	0422
Upper Sorbian	Germany	042E
Urdu	Islamic Republic of Pakistan	0420
Uzbek (Cyrillic)	Uzbekistan	0843
Uzbek (Latin)	Uzbekistan	0443
Vietnamese	Vietnam	042A
Welsh	United Kingdom	0452
Wolof	Senegal	0488
Yakut	Russia	0485
Yi	PRC	0478
Yoruba	Nigeria	046A

Platform-specific encoding and language IDs: Macintosh platform (platform ID = 1)

Macintosh encoding IDs (script manager codes)

The following encoding IDs are defined for use with the Macintosh platform:

Encoding ID	Script	Encoding ID	Script
0	Roman	17	Malayalam
1	Japanese	18	Sinhalese
2	Chinese (Traditional)	19	Burmese
3	Korean	20	Khmer
4	Arabic	21	Thai
5	Hebrew	22	Laotian
6	Greek	23	Georgian
7	Russian	24	Armenian
8	RSymbol	25	Chinese (Simplified)
9	Devanagari	26	Tibetan
10	Gurmukhi	27	Mongolian
11	Gujarati	28	Geez
12	Oriya	29	Slavic
13	Bengali	30	Vietnamese
14	Tamil	31	Sindhi
15	Telugu	32	Uninterpreted
16	Kannada		

Macintosh language IDs

The following are platform-specific language ID's assigned by Apple:

Language ID	Language	Language ID	Language
0	English	59	Pashto
1	French	60	Kurdish
2	German	61	Kashmiri
3	Italian	62	Sindhi
4	Dutch	63	Tibetan
5	Swedish	64	Nepali
6	Spanish	65	Sanskrit
7	Danish	66	Marathi
8	Portuguese	67	Bengali
9	Norwegian	68	Assamese
10	Hebrew	69	Gujarati
11	Japanese	70	Punjabi
12	Arabic	71	Oriya
13	Finnish	72	Malayalam
14	Greek	73	Kannada
15	Icelandic	74	Tamil
16	Maltese	75	Telugu
17	Turkish	76	Sinhalese
18	Croatian	77	Burmese
19	Chinese (Traditional)	78	Khmer
20	Urdu	79	Lao
21	Hindi	80	Vietnamese
22	Thai	81	Indonesian
23	Korean	82	Tagalong
24	Lithuanian	83	Malay (Roman script)
25	Polish	84	Malay (Arabic script)
26	Hungarian	85	Amharic
27	Estonian	86	Tigrinya
28	Latvian	87	Galla
29	Sami	88	Somali
30	Faroese	89	Swahili
31	Farsi/Persian	90	Kinyarwanda/Ruanda
32	Russian	91	Rundi

33	Chinese (Simplified)	92	Nyanja/Chewa
34	Flemish	93	Malagasy
35	Irish Gaelic	94	Esperanto
36	Albanian	128	Welsh
37	Romanian	129	Basque
38	Czech	130	Catalan
39	Slovak	131	Latin
40	Slovenian	132	Quenchua
41	Yiddish	133	Guarani
42	Serbian	134	Aymara
43	Macedonian	135	Tatar
44	Bulgarian	136	Uighur
45	Ukrainian	137	Dzongkha
46	Byelorussian	138	Javanese (Roman script)
47	Uzbek	139	Sundanese (Roman script)
48	Kazakh	140	Galician
49	Azerbaijani (Cyrillic script)	141	Afrikaans
50	Azerbaijani (Arabic script)	142	Breton
51	Armenian	143	Inuktitut
52	Georgian	144	Scottish Gaelic
53	Moldavian	145	Manx Gaelic
54	Kirghiz	146	Irish Gaelic (with dot above)
55	Tajiki	147	Tongan
56	Turkmen	148	Greek (polytonic)
57	Mongolian (Mongolian script)	149	Greenlandic
58	Mongolian (Cyrillic script)	150	Azerbaijani (Roman script)

Platform-specific encoding and language IDs: ISO platform (platform ID=2) [Deprecated]

The following encoding IDs are defined for use with ISO platform:

Code	ISO encoding
0	7-bit ASCII
1	ISO 10646
2	ISO 8859-1

There are no ISO-specific language IDs, and language-tag records are not supported on this platform. This means that it could potentially be used for encodings in the 'cmap' table but not for strings in the 'name' table. Note that use of the ISO platform in the 'cmap' table is deprecated.

Platform-specific encoding and language IDs: Custom platform (platform ID = 4)

ID	Custom encoding
0-255	OTF Windows NT compatibility mapping

In cases where a custom platform cmap is present for OTF Windows NT compatibility, the encoding ID must be set to the Windows charset value (in the range 0 to 255, inclusive) present in the .PFM file of the original Type 1 font. See the 'cmap' table for more details on the OTF Windows NT compatibility cmap.

There are no platform-specific language IDs defined for the Custom platform, and language-tag records are not supported on this platform. This means that it can be used for encodings in the 'cmap' table but not for strings in the 'name' table.

5.2.7.4 Name IDs

The following name IDs are pre-defined and they apply to all platforms unless indicated otherwise. Name IDs 23 to 255, inclusive, are reserved for future standard names. Name IDs 256 to 32767, inclusive, are reserved for font-specific names such as those referenced by a font's layout features.

Code	Meaning
0	Copyright notice.
1	Font Family name. This family name is assumed to be shared among fonts that differ only in weight or style (regular, italic, bold, bold). Font Family name is used in combination with Font Subfamily name (name ID 2). Some applications that use this pair of names assume that a Font Family name is shared by at most four fonts that form a font style-linking group: regular, italic, bold, and bold italic. To be compatible with the broadest range of platforms and applications, fonts should limit use of any given Font Family name in this manner. (This four-way distinction should also be reflected in OS/2.fsSelection bit settings.) For fonts within an extended typographic family that fall outside this four-way distinction, the distinguishing attributes should be reflected in the Font Family name so that those fonts appear as a separate font family. For example, the Font Family name for the Arial Narrow font is "Arial Narrow"; the Font Family name for the Arial Black font is "Arial Black". (Note that, in such cases, name ID 16 should also be included with a shared name that reflects the full, typographic family.)
2	Font Subfamily name. The Font Subfamily name distinguishes the font in a group with the same Font Family name (name ID 1). This is assumed to address style (italic, oblique) and weight variants only. A font with no distinctive weight or style (e.g. medium weight, not italic and OS/2.fsSelection bit 6 set) should have the string "Regular" as the Font Subfamily name (for English language). Font Subfamily name is used in combination with Font Family name (name ID 1). Some applications that use this pair of names assume that a Font Family name is shared by at most four fonts that form a font style-linking group. These four fonts may have Subfamily name values that reflect various weights or styles, with four-way "Bold" and "Italic" style-linking relationships indicated using OS/2.fsSelection bits 0, 5 and 6. Within an extended typographic family that includes fonts beyond regular, bold, italic, or bold italic, distinctions are made in the Font Family name, so that fonts appear to be in separate families. In some cases, this may lead to specifying a Subfamily name of "Regular" for a font that might not otherwise be considered a regular font. For example, the Arial Black font has a Font Family name of "Arial Black" and a Subfamily name of "Regular". (Note that, in such cases, name IDs 16 and 17 should also be included, using a shared value for name ID 16 that reflects the full typographic family, and values for name ID 17 that appropriately reflect the actual design variant of each font.)

3	Unique font identifier
4	Full font name that reflects all family and relevant subfamily descriptors. The full font name is generally a combination of name IDs 1 and 2, or of name IDs 16 and 17, or a similar human-readable variant. For fonts in extended typographic families (that is, families that include more than regular, italic, bold, and bold italic variants), values for name IDs 1 and 2 are normally chosen to provide compatibility with certain applications that assume a family has at most four style-linked fonts. In that case, some fonts may end up with a Subfamily name (name ID 2) of “Regular” even though the font would not be considered, typographically, a regular font. For such non-regular fonts in which name ID 2 is specified as “Regular”, the “Regular” descriptor would generally be omitted from name ID 4. For example, the Arial Black font has a Font Family name (name ID 1) of “Arial Black” and a Subfamily name (name ID 2) of “Regular”, but has a full font name (name ID 4) of “Arial Black”. Note that name IDs 16 and 17 should also be included in these fonts, and that name ID 4 would typically be a combination of name IDs 16 and 17, without needing any additional qualifications regarding “Regular”.
5	Version string. Should begin with the syntax 'Version <number>.<number>' (upper case, lower case, or mixed, with a space between "Version" and the number). The string must contain a version number of the following form: one or more digits (0-9) of value less than 65,535, followed by a period, followed by one or more digits of value less than 65,535. Any character other than a digit will terminate the minor number. A character such as ";" is helpful to separate different pieces of version information. The first such match in the string can be used by installation software to compare font versions. Some installers may require the string to start with "Version ", followed by a version number as above.
6	PostScript name for the font; Name ID 6 specifies a string which is used to invoke a PostScript language font that corresponds to this OFF font. When translated to ASCII, the name string must be no longer than 63 characters and restricted to the printable ASCII subset, codes 33-126, except for the 10 characters '[', ']', '(', ')', '{', '}', '<', '>', '\', '%'. In a CFF OT font, there is no requirement that this name be the same as the font name in the CFF's Name INDEX. Thus, the same CFF may be shared among multiple font components in an OFF Font Collection. See subclause 8.20 for additional information.
7	Trademark; this is used to save any trademark notice/information for this font. Such information should be based on legal advice. This is <i>distinctly</i> separate from the copyright.
8	Manufacturer Name.
9	Designer; name of the designer of the typeface.
10	Description; description of the typeface. Can contain revision information, usage recommendations, history, features, etc.
11	URL Vendor; URL of font vendor (with protocol, e.g., http://, ftp://). If a unique serial number is embedded in the URL, it can be used to register the font.
12	URL Designer; URL of typeface designer (with protocol, e.g., http://, ftp://).
13	License Description; description of how the font may be legally used, or different example scenarios for licensed use. This field should be written in plain language, not legalese.