
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 2015, Published in Switzerland

All rights reserved. Unless otherwise specified, 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
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	viii
Introduction.....	x
1 Scope	1
2 Normative references	2
3 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 Open font structure	5
4.5.1 Table directory	5
4.5.2 Calculating checksums.....	6
4.6 Font collections	6
4.6.1 The Font Collection file structure	7
4.6.2 TTC header	7
5 Open font tables	8
5.1 General	8
5.2 Required common tables.....	8
5.2.1 cmap – Character to glyph index mapping table	9
5.2.2 head – Font header.....	20
5.2.3 hhea – Horizontal header.....	22
5.2.4 hmtx – Horizontal metrics	23
5.2.5 maxp – Maximum profile.....	23
5.2.6 name – Naming table.....	24
5.2.7 OS/2 – Global font information table	43
5.2.8 Font class parameters - see informative Annex B for details.....	65
5.2.9 post – PostScript.....	65
5.3 TrueType outline tables	67
5.3.1 cvt – Control value table	68
5.3.2 fpgm – Font program	68
5.3.3 glyf – Glyf data.....	68
5.3.4 loca – Index to location.....	71
5.3.5 prep – Control value program	72
5.3.6 gasp – Grid-fitting and scan conversion procedure	72
5.4 PostScript outline tables	74
5.4.1 CFF – PostScript font program (Compact Font Format) table.....	74
5.4.2 VORG – Vertical origin table	74
5.5 Table for SVG glyph outlines	76
5.5.1 SVG – The SVG (Scalable Vector Graphics) table	76
5.5.2 Color Palettes	77
5.5.3 Glyph Identifiers	77
5.5.4 Glyph Semantics and Metrics	78
5.5.5 Glyph Rendering.....	78
5.6 Bitmap glyph tables	80
5.6.1 EBDT – Embedded bitmap data table.....	80
5.6.2 EBLC – Embedded bitmap location table	84
5.6.3 EBSC – Embedded bitmap scaling table	91
5.6.4 CBDT – Color bitmap data table	92

5.6.5	CBLC – Color bitmap location table	94
5.7	Optional tables	95
5.7.1	DSIG – Digital signature table	96
5.7.2	hdmx – Horizontal device metrics	98
5.7.3	kern – Kerning	99
5.7.4	LTSH – Linear threshold	102
5.7.5	PCLT – PCL 5 table	103
5.7.6	VDMX – Vertical device metrics	111
5.7.7	vhea – Vertical header table	114
5.7.8	vmtx – Vertical metric table	117
5.7.9	COLR – Color Table	119
5.7.10	CPAL – Palette Table	121
6	Advanced Open Font layout tables	123
6.1	Advanced Open Font layout extensions	123
6.1.1	Overview of advanced typographic layout extensions	123
6.1.2	TrueType versus OFF layout	125
6.1.3	OFF layout terminology	125
6.1.4	Text processing with OFF layout	128
6.2	OFF layout common table formats	129
6.2.1	Overview	129
6.2.2	Table organization	131
6.2.3	Scripts and languages	132
6.2.4	Features and lookups	134
6.2.5	Common table examples	143
6.3	Advanced typographic tables	152
6.3.1	BASE Baseline table	152
6.3.2	GDEF – The glyph definition table	173
6.3.3	GPOS – The glyph positioning table	186
6.3.4	GSUB – The glyph substitution table	246
6.3.5	JSTF – The justification table	286
6.3.6	MATH – The mathematical typesetting table	298
6.4	Layout tag registry	313
6.4.1	Scripts tags	313
6.4.2	Language tags	317
6.4.3	Feature tags	338
6.4.4	Baseline tags	402
7	Recommendations for OFF fonts	407
7.1	Byte ordering	407
7.2	'sfnt' version	407
7.3	Mixing outline formats	407
7.4	Filenames	407
7.5	Table alignment and length	407
7.6	First four glyphs in fonts	408
7.7	Shape of .notdef glyph	408
7.8	'BASE' table	408
7.9	'cmap' table	409
7.10	'cvt' table	409
7.11	'fpgm' table	409
7.12	'glyf' table	409
7.13	'hdmx' table	409
7.14	'head' table	410
7.15	'hhea' table	410
7.16	'hmtx' table	410
7.17	'kern' table	410
7.18	'loca' table	411
7.19	'LTSH' table	411
7.20	'maxp' table	411
7.21	'name' table	411
7.22	'OS/2' table	413

7.23	sTypoAscender, sTypoDescender and sTypoLineGap.....	413
7.24	'post' table.....	413
7.25	'prep' table.....	414
7.26	'VDMX' table.....	414
7.27	TrueType Collections.....	414
8	General recommendations.....	414
8.1	Optimized table ordering.....	414
8.2	Non-standard (Symbol) fonts.....	414
8.3	Device resolutions.....	415
8.4	Baseline to baseline distances.....	415
8.5	Style bits.....	416
8.6	Drop-out control.....	416
8.7	Embedded bitmaps.....	416
8.8	OFF CJK font guidelines.....	417
Annex A (informative) Patent Statements.....		418
Annex B (informative) Font Class and Font Subclass parameters.....		419
B.1	Introduction.....	419
B.2	sFamilyClass.....	419
B.3	Class ID=0 No Classification.....	419
B.4	Class ID=1 Oldstyle Serifs.....	419
B.4.1	Subclass ID = 0 : No Classification.....	419
B.4.2	Subclass ID = 1 : IBM Rounded Legibility.....	420
B.4.3	Subclass ID = 2 : Garalde.....	420
B.4.4	Subclass ID = 3 : Venetian.....	420
B.4.5	Subclass ID = 4 : Modified Venetian.....	420
B.4.6	Subclass ID = 5 : Dutch Modern.....	420
B.4.7	Subclass ID = 6 : Dutch Traditional.....	420
B.4.8	Subclass ID = 7 : Contemporary.....	420
B.4.9	Subclass ID = 8 : Calligraphic.....	420
B.4.10	Subclass ID = 9-14 : (reserved for future use).....	420
B.4.11	Subclass ID = 15 : Miscellaneous.....	421
B.5	Class ID=2 Transitional Serifs.....	421
B.5.1	Subclass ID = 0 : No Classification.....	421
B.5.2	Subclass ID = 1 : Direct Line.....	421
B.5.3	Subclass ID = 2 : Script.....	421
B.5.4	Subclass ID = 3-14 : (reserved for future use).....	421
B.5.5	Subclass ID = 15 : Miscellaneous.....	421
B.6	Class ID=3 Modern Serifs.....	421
B.6.1	Subclass ID = 0 : No Classification.....	421
B.6.2	Subclass ID = 1 : Italian.....	422
B.6.3	Subclass ID = 2 : Script.....	422
B.6.4	Subclass ID = 3-14 : (reserved for future use).....	422
B.6.5	Subclass ID = 15 : Miscellaneous.....	422
B.7	Class ID=4 Clarendon Serifs.....	422
B.7.1	Subclass ID = 0 : No Classification.....	422
B.7.2	Subclass ID = 1 : Clarendon.....	422
B.7.3	Subclass ID = 2 : Modern.....	422
B.7.4	Subclass ID = 3 : Traditional.....	422
B.7.5	Subclass ID = 4 : Newspaper.....	423
B.7.6	Subclass ID = 5 : Stub Serif.....	423
B.7.7	Subclass ID = 6 : Monotone.....	423
B.7.8	Subclass ID = 7 : Typewriter.....	423
B.7.9	Subclass ID = 8-14: (reserved for future use).....	423
B.7.10	Subclass ID = 15 : Miscellaneous.....	423
B.8	Class ID=5 Slab Serifs.....	423
B.8.1	Subclass ID = 0 : No Classification.....	423
B.8.2	Subclass ID = 1 : Monotone.....	423
B.8.3	Subclass ID = 2 : Humanist.....	424

B.8.4	Subclass ID = 3 : Geometric	424
B.8.5	Subclass ID = 4 : Swiss	424
B.8.6	Subclass ID = 5 : Typewriter.....	424
B.8.7	Subclass ID = 6-14 : (reserved for future use)	424
B.8.8	Subclass ID = 15 : Miscellaneous	424
B.9	Class ID=6 (reserved for future use).....	424
B.10	Class ID=7 Freeform Serifs	424
B.10.1	Subclass ID = 0 : No Classification	424
B.10.2	Subclass ID = 1 : Modern	425
B.10.3	Subclass ID = 2-14 : (reserved for future use)	425
B.10.4	Subclass ID = 15 : Miscellaneous	425
B.11	Class ID=8 Sans Serifs	425
B.11.1	Subclass ID = 0 : No Classification	425
B.11.2	Subclass ID = 1 : IBM Neo-grotesque Gothic.....	425
B.11.3	Subclass ID = 2 : Humanist.....	425
B.11.4	Subclass ID = 3 : Low-x Round Geometric	425
B.11.5	Subclass ID = 4 : High-x Round Geometric.....	425
B.11.6	Subclass ID = 5 : Neo-grotesque Gothic	426
B.11.7	Subclass ID = 6 : Modified Neo-grotesque Gothic	426
B.11.8	Subclass ID = 7-8 : (reserved for future use)	426
B.11.9	Subclass ID = 9 : Typewriter Gothic	426
B.11.10	Subclass ID = 10 : Matrix.....	426
B.11.11	Subclass ID = 11-14 : (reserved for future use)	426
B.11.12	Subclass ID = 15 : Miscellaneous	426
B.12	Class ID=9 Ornaments.....	426
B.12.1	Subclass ID = 0 : No Classification	426
B.12.2	Subclass ID = 1 : Engraver.....	426
B.12.3	Subclass ID = 2 : Black Letter	427
B.12.4	Subclass ID = 3 : Decorative.....	427
B.12.5	Subclass ID = 4 : Three Dimensional	427
B.12.6	Subclass ID = 5-14 : (reserved for future use)	427
B.12.7	Subclass ID = 15 : Miscellaneous	427
B.13	Class ID=10 Scripts	427
B.13.1	Subclass ID = 0 : No Classification	427
B.13.2	Subclass ID = 1 : Uncial	427
B.13.3	Subclass ID = 2 : Brush Joined	427
B.13.4	Subclass ID = 3 : Formal Joined	428
B.13.5	Subclass ID = 4 : Monotone Joined	428
B.13.6	Subclass ID = 5 : Calligraphic	428
B.13.7	Subclass ID = 6 : Brush Unjoined	428
B.13.8	Subclass ID = 7 : Formal Unjoined.....	428
B.13.9	Subclass ID = 8 : Monotone Unjoined	428
B.13.10	Subclass ID = 9-14 : (reserved for future use)	428
B.13.11	Subclass ID = 15 : Miscellaneous	428
B.14	Class ID=11 (reserved for future use).....	429
B.15	Class ID=12 Symbolic.....	429
B.15.1	Subclass ID = 0 : No Classification	429
B.15.2	Subclass ID = 1-2 : (reserved for future use)	429
B.15.3	Subclass ID = 3 : Mixed Serif	429
B.15.4	Subclass ID = 4-5 : (reserved for future use)	429
B.15.5	Subclass ID = 6 : Oldstyle Serif	429
B.15.6	Subclass ID = 7 : Neo-grotesque Sans Serif	429
B.15.7	Subclass ID = 8-14 : (reserved for future use)	429
B.15.8	Subclass ID = 15 : Miscellaneous	430
B.16	Class ID=13 Reserved	430
B.17	Class ID=14 Reserved	430
Annex C	(informative) Earlier versions of OS/2 – OS/2 and Windows metrics	431
C.1	OS/2 - OS/2 and Windows metrics (version 0).....	431
C.2	OS/2 - OS/2 and Windows metrics (version 1).....	450

C.3 OS/2 - OS/2 and Windows metrics (version 2) 471

C.4 OS/2 - OS/2 and Windows metrics (version 3) 490

C.5 OS/2 - OS/2 and Windows metrics (version 4) 509

Annex D (informative) OFF Mirroring Pairs List.....531

Annex E (normative) Registration of Media Type: application/font-sfnt 538

Bibliography.....541

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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).

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: [Foreword — Supplementary information](#).

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC 14496-22:2009), which has been technically revised. It also incorporates ISO/IEC 14496-22:2009/Cor 1:2010, ISO/IEC 14496-22:2009/Amd 1:2010 and ISO/IEC 14496-22:2009/Amd 2:2010.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

- *Part 1: Systems*
- *Part 2: Visual*
- *Part 3: Audio*
- *Part 4: Conformance testing*
- *Part 5: Reference software*
- *Part 6: Delivery Multimedia Integration Framework (DMIF)*
- *Part 7: Optimized reference software for coding of audio-visual objects*
- *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

- *Part 9: Reference hardware description*
- *Part 10: Advanced Video Coding*
- *Part 11: Scene description and application engine*
- *Part 12: ISO base media file format*
- *Part 13: Intellectual Property Management and Protection (IPMP) extensions*
- *Part 14: MP4 file format*
- *Part 15: Carriage of network abstraction layer (NAL) unit structured video in ISO base media file format*
- *Part 16: Animation Framework eXtension (AFX)*
- *Part 17: Streaming text format*
- *Part 18: Font compression and streaming*
- *Part 19: Synthesized texture stream*
- *Part 20: Lightweight Application Scene Representation (LAsER) and Simple Aggregation Format (SAF)*
- *Part 21: MPEG-J Graphics Framework eXtensions (GFX)*
- *Part 22: Open Font Format*
- *Part 23: Symbolic Music Representation*
- *Part 24: Audio and systems interaction*
- *Part 25: 3D Graphics Compression Model*
- *Part 26: Audio conformance*
- *Part 27: 3D Graphics conformance*
- *Part 28: Composite Font Representation*
- *Part 29: Web video coding*
- *Part 30: Timed text and other visual overlays in ISO base media file format*

The following part is in preparation:

- *Part 30: Video coding for browsers*

A future part dealing with Internet video coding is planned.

Introduction

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 a patent.

The ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the companies listed in Annex A.

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 in Annex A. 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 part of ISO/IEC 14496 specifies the Open Font Format (OFF) specification, the TrueType™ⁱ and Compact Font Format (CFF) outline formats, and the TrueType hinting language. Many references to both TrueType and PostScript exist throughout this document, as Open Font Format fonts combine the two technologies.

NOTE This specification is based on the OpenType®ⁱⁱ font format specification, and is technically equivalent to that specification.

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 is an extension of the TrueType font format, adding support for PostScript 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) 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

PostScript®ⁱⁱⁱ 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.

2 Normative references

The following referenced documents are indispensable for the application 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 15948:2004, *Information technology — Computer graphics and image processing — Portable Network Graphics: Functional specification (also available as W3C Recommendation[15])*

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 7.0, <<http://www.unicode.org/versions/Unicode7.0.0/>>

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

3 Abbreviated terms

List of abbreviated terms.

ACF	Average Character Face
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ATM	Adobe Type Manager
BMP	[Unicode] Basic Multilingual Plane (also known as UCS-2)
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
PRC	People's Republic of China
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 PostScript 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.

4.2 Filenames

For filename extension conventions, see the "Filenames" section in clause 7 "Recommendations".

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
BYTE	8-bit unsigned integer.
CHAR	8-bit signed integer.
USHORT	16-bit unsigned integer.
SHORT	16-bit signed integer.
UINT24	24-bit unsigned integer.
ULONG	32-bit unsigned integer.

LONG	32-bit signed integer.
Fixed	32-bit signed fixed-point number (16.16)
FUNIT	Smallest measurable distance in the em space.
FWORD	16-bit signed integer (SHORT) that describes a quantity in FUnits.
UFWORD	16-bit unsigned integer (USHORT) that describes a quantity in FUnits.
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 script, language system, feature, or baseline
GlyphID	Glyph index number, same as uint16 (length = 16 bits)
Offset	Offset to a table, same as uint16 (length = 16 bits), NULL Offset = 0x0000

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

4.4 Table version numbers

Most tables have version numbers, and the version number for the entire font is contained in the Table Directory. It should be noted that there are two different table version number types, each with its own numbering scheme. USHORT version numbers always start at zero (0). Fixed version numbers start at one (1.0 or 0x00010000), except where noted (EBDT, EBLC and EBSC tables).

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 reject newer versions gracefully, if the changes are incompatible.

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. Tables with non-zero minor version numbers always specify the literal value of the version number since the normal representation of Fixed numbers is not necessarily followed. For example, the version number of 'maxp' table version 0.5 is 0x00005000, and that of 'vhea' table version 1.1 is 0x00011000. If an implementation understands a major version number, then it can safely proceed reading the table. The minor version number indicates extensions to the format that are undetectable by implementations that do not support them.

The only exception to this is the Offset Table's sfnt version. This serves solely to identify whether the OFF font contains TrueType outlines (a value of 1.0) or CFF data (the tag 'OTTO'), as described in subclause 4.5, 'Open Font Structure.'

When a USHORT number is used to indicate version, it should be treated as though it were a minor version number; i.e., all format changes are compatible extensions.

4.5 Open font structure

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 TrueType collection, the beginning point of the Offset Table for each font is indicated in the TTCHheader.

Offset Table		
Type	Name	Description
Fixed	sfnt version	0x00010000 for version 1.0 or 'OTTO'.
USHORT	numTables	Number of tables.
USHORT	searchRange	(Maximum power of 2 $\leq$ numTables) x 16.
USHORT	entrySelector	Log2(maximum power of 2 $\leq$ numTables).
USHORT	rangeShift	NumTables x 16-searchRange.

OFF fonts that contain TrueType outlines should use the value of 1.0 for the sfnt version. OFF fonts containing CFF data should use the tag 'OTTO' as the sfnt version number.

4.5.1 Table directory

The Offset Table is followed immediately by the Table Record entries. Entries in the Table Record must 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
ULONG	tag	4 -byte identifier.
ULONG	checksum	Checksum for this table.
ULONG	Offset	Offset from beginning of TrueType font

		file.
ULONG	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.

Tags are the names given to tables in the OFF font file. All tag names consist of four characters. Names with less than four letters are allowed if followed by the necessary trailing spaces. All tag names defined within a font (e.g., table names, feature tags, language tags) must be built from printing characters represented by ASCII values 32-126.

NOTE Tag names are case sensitive.

4.5.2 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:

```

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

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

```

NOTE This function implies that the length of a table must be 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 should 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 the hex value B1B0AFBA.
- 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

A Font Collection (either TTC or OTC, formerly known as TrueType Collection) is a means of delivering multiple OFF fonts 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, 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 the case. TTC files may contain either TrueType or CFF outlines, 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 should be understood that it is used for both CFF and TrueType outlines

4.6.1 The Font Collection file structure

The TTC File Structure

A TrueType Collection file consists of a single TTC Header table, one or more Offset Tables with Table Directories, and a number of OFF tables. The TTC Header must be located at the beginning of the TTC file.

The TTC file must contain a complete Offset Table and Table Directory for each font. 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.

4.6.2 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 both CFF and TrueType, for backward compatibility)
FIXED	Version	Version of the TTC Header (1.0), 0x00010000
ULONG	numFonts	Number of fonts in TTC
ULONG	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'
FIXED	Version	Version of the TTC Header (2.0), 0x00020000
ULONG	numFonts	Number of fonts in TTC
ULONG	OffsetTable[numFonts]	Array of Offsets to the OffsetTable for each font from the beginning of the file
ULONG	ulDsigTag	Tag indicating that a DSIG table exists, 0x44534947 ('DSIG') (null if no signature)
ULONG	ulDsigLength	The length (in bytes) of the DSIG table (null if no signature)
ULONG	ulDsigOffset	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

Whether TrueType or PostScript 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

5.2.1 cmap – Character to glyph index mapping table

5.2.1.1 Table structure

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. 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.

The table header indicates the character encodings for which subtables are present. Each subtable is in one of seven possible formats and begins with a format code indicating the format used.

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

cmap Header

Type	Name	Description
USHORT	Version	Table version number (0)
USHORT	numTables	Number of encoding tables that follow

The cmap table header is followed by an array of encoding records that specify the particular encoding and the Offset to the subtable for that encoding. The number of encoding records is *numTables*. An encoding record entry looks like:

Encoding Record

Type	Name	Description
USHORT	platformID	Platform ID.
USHORT	encodingID	Platform-specific encoding ID.
ULONG	Offset	Byte Offset from beginning of table to the subtable for this encoding

NOTE The values of platformID, encodingID and languageID are defined in the 'name' table description in subclause 4.2.6.

The platform ID and platform-specific encoding ID in the header entry (and, in the case of the Macintosh platform, the language field in the subtable itself) are used to specify a particular 'cmap' encoding. The header entries 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.

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

All Windows Unicode BMP encodings (Platform ID = 3, Encoding ID = 1) must provide at least a Format 4 'cmap' subtable. If the font is meant to support supplementary (non-BMP) Unicode characters, it will additionally need a Format 12 subtable with a platform encoding ID 10. The contents of the Format 12 subtable need to be a superset of the contents of the Format 4 subtable. It is strongly recommended using a BMP Unicode 'cmap' for all fonts. However, some other encodings that appear in current fonts follow:

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

Unicode Variation Sequences supported by the font may be specified in the 'cmap' table under platform ID 0 and encoding ID 5, using a format 14 cmap subtable specified in the subclause 4.2.1.3.8.

5.2.1.2 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: (a) 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; (b) add Windows character set (CharSet 0) to the list of CharSets supported by the font; and (c) 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 This 'cmap' subtable must use Format 0 or 6 for its subtable, and the encoding must be identical to the CFF's encoding.

5.2.1.3 cmap subtable formats

Note on 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.

5.2.1.3.1 Format 0: Byte encoding table

This is the Macintosh platform standard character to glyph index mapping table which is available via the informative reference [7] in the bibliography.

Type	Name	Description
USHORT	format	Format number is set to 0.
USHORT	length	This is the length in bytes of the subtable.
USHORT	language	Please see "Note on the language field in 'cmap' subtables" in this document.
BYTE	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.

5.2.1.3.2 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.

Type	Name	Description
USHORT	format	Format number is set to 2.
USHORT	length	This is the length in bytes of the subtable.
USHORT	language	Please see "Note on the language field in 'cmap' subtables" in this document.
USHORT	subHeaderKeys[256]	Array that maps high bytes to subHeaders: value is subHeader index * 8.
4 words struct	subHeaders[]	Variable-length array of subHeader structures.
USHORT	glyphIndexArray[]	Variable-length array containing subarrays used for mapping the low byte of 2-byte characters.

A subHeader is structured as follows:

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

5.2.1.3.3 Format 4: Segment mapping to delta values

This is the Windows standard character to glyph index mapping table for fonts that support Unicode ranges other than the range [U+D800 - U+DFFF] (defined as Surrogates Area, in the Unicode Standard) which is used for UCS-4 characters. If a font supports this character range (i.e. in turn supports the UCS-4 characters) a subtable in this format with a platform specific encoding ID 1 is yet needed, in addition to a subtable in format 12 with a platform specific encoding ID 10. Please see details on format 12 below, for fonts that support UCS-4 characters on Windows.

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 must occur in the following order:

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

Type	Name	Description
USHORT	format	Format number is set to 4.
USHORT	length	This is the length in bytes of the subtable.
USHORT	language	Please see "Note on the language field in 'cmap' subtables" in this document.
USHORT	segCountX2	2 x segCount.
USHORT	searchRange	$2 \times (2^{\lceil \log_2(\text{segCount}) \rceil})$
USHORT	entrySelector	$\log_2(\text{searchRange}/2)$
USHORT	rangeShift	$2 \times \text{segCount} - \text{searchRange}$
USHORT	endCode[segCount]	End characterCode for each segment, last=0xFFFF.
USHORT	reservedPad	Set to 0.
USHORT	startCode[segCount]	Start character code for each segment.
SHORT	idDelta[segCount]	Delta for all character codes in segment.
USHORT	idRangeOffset[segCount]	Offsets into glyphIdArray or 0
USHORT	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 * largest power of 2 <=39)
entrySelector	5	$\log_2(32)$
rangeShift	14	$2 \times 39 - 64$

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 must 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:

```
*(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
...and so on.
```

It should be noted that the delta values could be reworked so as to reorder the segments.

5.2.1.3.4 Format 6: Trimmed table mapping

Type	Name	Description
USHORT	format	Format number is set to 6.

USHORT	length	This is the length in bytes of the subtable.
USHORT	language	Please see "Note on the language field in 'cmap' subtables" in this document.
USHORT	firstCode	First character code of subrange.
USHORT	entryCount	Number of character codes in subrange.
USHORT	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.

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

Format 8 is a bit like format 2, in that it provides for mixed-length character codes. If a font contains characters from the Unicode Surrogates Area (U+D800-U+DFFF), which are UCS-4 characters; it's likely that it will also include other, regular 16-bit Unicodes as well. We therefore need a format to map a mixture of 16-bit and 32-bit character codes, just as format 2 allows a mixture of 8-bit and 16-bit 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. 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.

Here's the format 8 subtable format:

Type	Name	Description
USHORT	format	Subtable format; set to 8.
USHORT	reserved	Reserved; set to 0
ULONG	length	Byte length of this subtable (including the header)
ULONG	language	Please see "Note on the language field in 'cmap' subtables" in this document.
BYTE	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
ULONG	nGroups	Number of groupings which follow

Here follow the individual groups. Each group has the following format:

Type	Name	Description
------	------	-------------

ULONG	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
ULONG	endCharCode	Last character code in this group; same condition as listed above for the startCharCode
ULONG	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 must be sorted by increasing startCharCode. A group's endCharCode must 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 might work advantageously on some platforms for non-Unicode encodings, Windows does not support it for Unicode encoded UCS-4 characters.

5.2.1.3.6 Format 10: Trimmed array

Format 10 is a bit like format 6, in that it defines a trimmed array for a tight range of 32-bit character codes:

Type	Name	Description
USHORT	format	Subtable format; set to 10.
USHORT	reserved	Reserved; set to 0
ULONG	length	Byte length of this subtable (including the header)
ULONG	language	Please see "Note on the language field in 'cmap' subtables" in this document.
ULONG	startCharCode	First character code covered
ULONG	numChars	Number of character codes covered
USHORT	glyphs[]	Array of glyph indices for the character codes covered

5.2.1.3.7 Format 12: Segmented coverage

This is the Windows platform standard character to glyph index mapping table for fonts supporting the UCS-4 characters in the Unicode Surrogates Area (U+D800 - U+DFFF). It is a bit like format 4, in that it defines segments for sparse representation in 4-byte character space. Here's the subtable format:

Type	Name	Description
USHORT	format	Subtable format; set to 12.
USHORT	reserved	Reserved; set to 0
ULONG	length	Byte length of this subtable (including the header)
ULONG	language	Please see "Note on the language field in 'cmap' subtables" in this document.
ULONG	nGroups	Number of groupings which follow

Fonts providing Unicode encoded UCS-4 character support for Windows 2000 and later, need to have a subtable with platform ID 3, platform specific encoding ID 1 in format 4; and in addition, need to have a subtable for platform ID 3, platform specific encoding ID 10 in format 12. Please note that the content of format 12 subtable, needs to be a super set of the content in the format 4 subtable. The format 4 subtable needs to be in the cmap table to enable backward compatibility needs.

Here follow the individual groups, each of which has the following format:

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

Groups must be sorted by increasing startCharCode. A group's endCharCode must 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.

5.2.1.3.8 Format 13: Many-to-one range mappings

This subtable deals with situations where 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 (see also subclause 5.2.2, Font header table, flags: bit 14).

Type	Name	Description
USHORT	format	Subtable format; set to 13.
USHORT	reserved	Reserved; set to 0
ULONG	length	Byte length of this subtable (including the header)

ULONG	language	Please see "Note on the language field in 'cmap' subtables" in this document.
ULONG	nGroups	Number of groupings which follow

Here follow the individual groups, each of which has the following format:

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

NOTE: the subtable formats 12 and 13 have the same data structure; they differ only in the interpretation of the startGlyphID/glyphID fields.

5.2.1.3.9 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 UCS-4 or the BMP cmap 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.

Format 14 header		
Type	Name	Description
USHORT	format	Subtable format. Set to 14.
ULONG	length	Byte length of this subtable (including this header)
ULONG	numVarSelectorRecords	Number of variation Selector Records

This is immediately followed by 'numVarSelectorRecords' Variation Selector Records.

Variation Selector Record		
Type	Name	Description
UINT24	varSelector	Variation selector
ULONG	defaultUVSOffset	Offset to Default UVS Table. May be 0.
ULONG	nonDefaultUVSOffset	Offset 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'. All offsets in a record are relative to the beginning of the format 14 cmap subtable.

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
ULONG	numUnicodeValueRanges	Number of ranges that follow

This is immediately followed by 'numUnicodeValueRanges' Unicode Value Ranges, each of which represents a contiguous range of Unicode values.

Unicode Value Range		
Type	Name	Description
UINT24	startUnicodeValue	First value in this range
BYTE	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') must 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
ULONG	numUVSMappings	Number of UVS Mappings that follow

This is immediately followed by 'numUVSMappings' UVS Mappings.

UVS Mapping		
Type	Name	Description
UINT24	unicodeValue	Base Unicode value of the UVS
USHORT	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

5.2.2 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
Fixed	Table version number	0x00010000 for version 1.0.
Fixed	fontRevision	Set by font manufacturer.
ULONG	checksumAdjustment	To compute: set it to 0, sum the entire font as ULONG, then store 0xB1B0AFBA - sum.
ULONG	magicNumber	Set to 0x5F0F3CF5.
USHORT	flags	Bit 0: Baseline for font at y=0; Bit 1: Left sidebearing point at x=0; 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); Bits 5-10: These bits are not defined in OFF Bit 11: Font data is 'lossless' as a results of having been subjected to optimizing transformation and/or compression (such as e.g. compression mechanisms defined by ISO/IEC 14496-18, MicroType® Express, WOFF 2.0 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 ClearType®. Note, fonts that rely on embedded bitmaps (EBDT) for rendering should not be considered optimized for ClearType, and therefore 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
USHORT	unitsPerEm	Valid range is from 16 to 16384. This value should be a power of 2 for fonts that have TrueType outlines.
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
SHORT	xMin	For all glyph bounding boxes.
SHORT	yMin	For all glyph bounding boxes.
SHORT	xMax	For all glyph bounding boxes.
SHORT	yMax	For all glyph bounding boxes.
USHORT	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).
USHORT	lowestRecPPEM	Smallest readable size in pixels.
SHORT	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
SHORT	indexToLocFormat	0 for short Offsets, 1 for long.
SHORT	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.

5.2.3 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 must be set to 0.

Type	Name	Description
Fixed	Table version number	0x00010000 for version 1.0.
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 Windows 3.1, System 6, and System 7.
UFWORD	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	$\text{Max}(\text{lsb} + (\text{xMax} - \text{xMin}))$.
SHORT	caretSlopeRise	Used to calculate the slope of the cursor (rise/run); 1 for vertical.
SHORT	caretSlopeRun	0 for vertical.
SHORT	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
SHORT	(reserved)	set to 0
SHORT	(reserved)	set to 0
SHORT	(reserved)	set to 0
SHORT	(reserved)	set to 0

SHORT	metricDataFormat	0 for current format.
USHORT	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.

5.2.4 hmtx – Horizontal metrics

The type longHorMetric is defined as an array where each element has two parts: the advance width, which is of type USHORT, and the left side bearing, which is of type SHORT. These fields are in font design units.

```
typedef struct _longHorMetric {
    USHORT advanceWidth;
    SHORT   lsb;
} longHorMetric;
```

Field	Type	Description
hMetrics	longHorMetric [numberOfHMetrics]	Paired advance width and left side bearing values for each glyph. The value numberOfHMetrics comes from the 'hhea' table. If the font is monospaced, only one entry need be in the array, but that entry is required. The last entry applies to all subsequent glyphs.
leftSideBearing	SHORT[]	Here the advanceWidth is assumed to be the same as the advanceWidth for the last entry above. The number of entries in this array is derived from numGlyphs (from 'maxp' table) minus numberOfHMetrics. This generally is used with a run of monospaced glyphs (e.g., Kanji fonts or Courier fonts). Only one run is allowed and it must be at the end. This allows a monospaced font to vary the left side bearing values for each glyph.

A font rendering engine must use the advanceWidths in the hmtx table for the advances of a CFF OFF font, even though the CFF table specifies its own glyph widths. Note that fonts in a Font Collection which share a CFF table may specify different advanceWidths in their hmtx table for a particular glyph index.

For any glyph, xmax and xmin are given in 'glyf' table, lsb and aw are given in 'hmtx' table. rsb is calculated as follows:

$$rsb = aw - (lsb + xmax - xmin)$$

If pp1 and pp2 are phantom points used to control lsb and rsb, their initial position in x is calculated as follows:

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

5.2.5 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	Table number	version	0x00005000 for version 0.5 (It should be noted that the difference in the representation of a non-zero fractional part, in Fixed numbers.)
USHORT	numGlyphs		The number of glyphs in the font.

Version 1.0

Type	Name	Description
Fixed	Table version number	0x00010000 for version 1.0.
USHORT	numGlyphs	The number of glyphs in the font.
USHORT	maxPoints	Maximum points in a non-composite glyph.
USHORT	maxContours	Maximum contours in a non-composite glyph.
USHORT	maxCompositePoints	Maximum points in a composite glyph.
USHORT	maxCompositeContours	Maximum contours in a composite glyph.
USHORT	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.
USHORT	maxTwilightPoints	Maximum points used in Z0.
USHORT	maxStorage	Number of Storage Area locations.
USHORT	maxFunctionDefs	Number of FDEFs, equals to the highest function number + 1.
USHORT	maxInstructionDefs	Number of IDEFs.
USHORT	maxStackElements	Maximum stack depth ² .
USHORT	maxSizeOfInstructions	Maximum byte count for glyph instructions.
USHORT	maxComponentElements	Maximum number of components referenced at "top level" for any composite glyph.
USHORT	maxComponentDepth	Maximum levels of recursion; 1 for simple components.

²This includes Font and CV Programs, as well as the instructions for each glyph.

5.2.6 name – Naming table

5.2.6.1 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 then refer to them simply by their index number. Clients that need a particular string can look it up by its platform ID, character encoding ID, language ID

and name ID. It should be noted that some platforms may require single byte character strings, while others may require double byte strings.

For historical reasons, some applications which install fonts perform version control using Macintosh platform (platform ID 1) strings from the 'name' table. Because of this, it is strongly recommended that the 'name' table of all fonts include Macintosh platform strings and that the syntax of the version number (name id 5) follows the guidelines given in this document.

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.

The format 0 Naming Table is organized as follows:

Type	Name	Description
USHORT	format	Format selector (=0).
USHORT	count	Number of name records.
USHORT	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.

Each string in the string storage is referenced by a name record. As described below, each name record is keyed using four numeric identifiers. Two of these are a platform identifier and a language identifier. When used in a format 0 Naming table, the language identifiers must be less than 0x8000 and are given platform-specific interpretations. (Exceptions to that limit may be made in the case of user-defined platforms, however.) Complete details of name records, the various identifiers and their interpretations are provided below.

The format 1 Naming Table adds additional elements, as follows:

Type	Name	Description
USHORT	format	Format selector (=1).
USHORT	count	Number of name records.
USHORT	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.
USHORT	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 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
USHORT	length	Language-tag string length (in bytes).
USHORT	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 must conform to IETF specification BCP 47. This provides tags such as "en", "fr-CA" and "zh-Hant" to identify languages, including dialects, written form and other variations.

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 identify 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.

Name records

Each *NameRecord* looks like this:

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

Following are the descriptions of the four kinds of ID. The specific values listed here are predefined; new ones may be added in the future. Similar to the character encoding table, the NameRecords are sorted by platform ID, then platform-specific ID, then language ID, and then by name ID.

5.2.6.2 Platform IDs, Platform-specific encoding IDs and Language IDs

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. Values greater than or equal to

0x8000 are associated with language-tag records, as described above. Not all platforms have platform-specific language IDs, and not all platforms support language-tag records.

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

It should be noted that platform ID 2 (ISO) has been deprecated as of OpenType v1.3. It was intended to represent ISO/IEC 10646, as opposed to Unicode; both standards have identical character code assignments.

Platform ID values 240 through 255 are reserved for user-defined platforms. This specification will never assign these values to a registered platform.

Unicode platform-specific encoding IDs (platform ID = 0)

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.

Windows platform-specific encoding IDs (platform ID= 3)

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

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. 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 Language IDs (platform ID = 3)

The language ID (LCID in the table below) refers to a value which identifies the language in which a particular string is written. The platform-specific Language IDs assigned by Microsoft are listed below. Any platform-specific language IDs must be less than 0x8000. Language IDs greater than or equal to 0x8000 may be used together with language-tag records.

Primary Language	Region	LCID
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

**Macintosh platform-specific encoding IDs
(script manager codes), (platform ID = 1)**

Code	Script	Code	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 (platform ID = 1)

The platform-specific language ID's assigned by Apple are listed below. Any platform-specific language IDs must be less than 0x8000. Language IDs greater than or equal to 0x8000 may be used together with language-tag records.

Code	Language	Code	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	Tagalog
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	14	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)

ISO specific encodings (platform ID=2) [deprecated]

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 can be used for encodings in the 'cmap' table but not for strings in the 'name' table.

Custom platform-specific encoding IDs (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.

5.2.6.3 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. Up to four fonts can share the Font Family name, forming a font style linking group (regular, italic, bold, bold italic - as defined by OS/2.fsSelection bit settings).
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 (light, bold, black, etc.). A font with no particular differences in weight or style (e.g. medium weight, not italic and fsSelection bit 6 set) should have the string "Regular" stored in this position.

3	Unique font identifier
4	Full font name; a combination of strings 1 and 2, or a similar human-readable variant. If string 2 is "Regular", it is sometimes omitted from name ID 4.
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 '[', ']', '(', ')', '{', '}', '<', '>', '/', '%'. See 'name' table section of clause 7 "Recommendations for OFF fonts" 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.
14	License Info URL; URL where additional licensing information can be found.
15	Reserved.
16	Typographic Family name: The typographic family grouping doesn't impose any constraints on the number of faces within it, in contrast with the 4-style family grouping (ID 1), which is present both for historical reasons and to express style linking groups. If name ID 16 is absent, then name ID 1 is considered to be the typographic family name. (In earlier versions of the specification, name ID 16 was known as "Preferred Family".)
17	Typographic Subfamily name: This allows font designers to specify a subfamily name within the typographic family grouping. This string must be unique within a particular typographic family. If it is absent, then name ID 2 is considered to be the typographic subfamily name. (In earlier versions of the specification, name ID 17 was known as "Preferred Subfamily".)

- 18 Compatible Full (Macintosh only); On the Macintosh, the menu name is constructed using the FOND resource. This usually matches the Full Name. If you want the name of the font to appear differently than the Full Name, you can insert the Compatible Full Name in ID 18.
- 19 Sample text; This can be the font name, or any other text that the designer thinks is the best sample to display the font in.
- 20 PostScript CID findfont name; Its presence in a font means that the nameID 6 holds a PostScript font name that is meant to be used with the "composefont" invocation in order to invoke the font in a PostScript interpreter. See the definition of name ID 6.
- The value held in the name ID 20 string is interpreted as a PostScript font name that is meant to be used with the "findfont" invocation, in order to invoke the font in a PostScript interpreter.
- When translated to ASCII, this name string must be restricted to the printable ASCII subset, codes 33 through 126, except for the 10 characters: '[', ']', '(', ')', '{', '}', '<', '>', '/', '%'. See 'name' table section of clause 7 "Recommendations for OFF fonts" for additional information.
- 21 WWS family name (see OS/2 fsSelection field for details). If bit 8 of 'fsSelection' field is set, the font belongs to WWS font families that are composed of font faces that differ only in Weight, Width and Slope. Non-WWS font families may contain faces for weight, width and slope, in addition to faces for other traditional attributes such as "handwriting", "caption", "subheading", "display", "optical" etc. This ID may define the additional attributes of non-WWS font families. Examples of name ID 21: "Minion Pro Caption" and "Minion Pro Display". (Name ID 16 would be "Minion Pro" for these examples.)
- 22 WWS subfamily name; Should be similar to ID 21, but reflect only weight, width and slope attributes of the font. Examples of name ID 22: "Semibold Italic", "Bold Condensed". (Name ID 17 could be "Semibold Italic Caption", or "Bold Condensed Display", for example.)

NOTE While both Apple and Microsoft support the same set of name strings, the interpretations may be somewhat different. But since name strings are stored by platform, encoding and language (placing separate strings for both Apple and MS platforms), this should not present a problem.

The key information for this table for MS fonts relates to the use of strings 1, 2 and 4. To better help understand, some examples of name usage, weight class and style flags have been created.

The following is an example of how name strings would be made for the Arial family:

Font	Name ID 1	Name ID 2	Name ID 4	Name ID 16	Name ID 17
Arial Narrow	Arial Narrow	Regular	Arial Narrow	Arial	Narrow
Arial Narrow Italic	Arial Narrow	Italic	Arial Italic Narrow	Arial	Narrow Italic
Arial Narrow Bold	Arial Narrow	Bold	Arial Bold Narrow	Arial	Narrow Bold
Arial Narrow Bold Italic	Arial Narrow	Bold Italic	Arial Bold Italic Narrow	Arial	Narrow Bold Italic
Arial	Arial	Regular	Arial	Arial	

Arial Italic	Arial	Italic	Arial Italic	Arial	Italic
Arial Bold	Arial	Bold	Arial Bold	Arial	Bold
Arial Bold Italic	Arial	Bold Italic	Arial Bold Italic	Arial	Bold Italic
Arial Black	Arial Black	Regular	Arial Black	Arial	Black
Arial Black Italic	Arial Black	Italic	Arial Black Italic	Arial	Black Italic

In addition to name strings, OS/2.usWeightClass, OS/2.usWidthClass, OS/2.fsSelection style bits, and head.macStyle bits are shown. These settings allow the fonts to fit together into a single family of varying weight and compression/expansion.

Font	OS/2 usWeight Class	OS/2 usWidthClass	OS/2 fsSelection Italic	OS/2 fsSelection n Bold	OS/2 fsSelection Regular	head macStyle Bold	head macStyle Italic	head macStyle Condensed	head macStyle Extended
Arial Narrow	400	3			x			x	
Arial Narrow Italic	400	3	x				x	x	
Arial Narrow Bold	700	3		x		x		x	
Arial Narrow Bold Italic	700	3	x	x		x	x	x	
Arial	400	5			x				
Arial Italic	400	5	x				x		
Arial Bold	700	5		x		x			
Arial Bold Italic	700	5	x	x		x	x		
Arial Black	900	5							
Arial Black Italic	900	5	x				x		