The OpenType Font File

Editor's note: *The following is a draft for a revision of the OpenType Font File chapter for OT 1.9, introducing version 1 of the COLR table. Changes shown here reflect changes from version 1.8.3 of the OT spec.*

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

Filenames

OpenType fonts may have the extension .OTF, .TTF, .OTC or .TTC. The extensions .OTC and .TTC should only be used for font collection files. For additional information on filename extension conventions, see "Filenames" in Recommendations for OpenType fonts.

Data Types

The following data types are used in the OpenType font file. All OpenType fonts use Motorola-style byte ordering (Big Endian):

Data Type	Description
uint8	8-bit unsigned integer.
int8	8-bit signed integer.
uint16	16-bit unsigned integer.
int16	16-bit signed integer.
uint24	24-bit unsigned integer.
uint32	32-bit unsigned integer.
int32	32-bit signed integer.
Fixed	32-bit signed fixed-point number (16.16)
FWORD	int16 that describes a quantity in font design units.
UFWORD	uint16 that describes a quantity in font design units.
F2DOT14	16-bit signed fixed number with the low 14 bits of fraction (2.14).
LONGDATETIME	Date represented in number of seconds since 12:00 midnight, January 1, 1904. The value is represented as a signed 64-bit integer.
Tag	Array of four uint8s (length = 32 bits) used to identify a table, design-variation axis, script, language system, feature, or baseline

Offset16	Short offset to a table, same as uint16, NULL offset = 0x0000
Offset24	24-bit offset to a table, same as uint24, NULL offset = 0x000000
Offset32	Long offset to a table, same as uint32, NULL offset = 0x00000000

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

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

A Tag value is a uint8 array. Each byte within the array must have a value in the range 0x20 to 0x7E. This corresponds to the range of values of Unicode Basic Latin characters in UTF-8 encoding, which is the same as the printable ASCII characters. As a result, a Tag value can be re-interpreted as a four-character sequence, which is conventionally how they are referred to. Formally, however, the value is a byte array. When re-interpreted as characters, the Tag value is case sensitive. It must have one to four non-space characters, padded with trailing spaces (byte value 0x20). A space character cannot be followed by a non-space character.

Table Version Numbers

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

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

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

The Table Directory uses a uint32 field with an enumeration of defined values that represent four-character tags; see the section below, “Organization of an OpenType Font”, for details.

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

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

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

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

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

Organization of an OpenType Font

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

The OpenType 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 an OpenType Font Collection file (see below), the beginning point of the Offset Table for each font is indicated in the TTCHeader.

Offset Table:

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

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

> *Note:* The Apple specification for TrueType fonts allows for 'true' and 'typ1' for sfnt version. These version tags should not be used for fonts which contain OpenType tables.

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
Tag	tableTag	Table identifier.
uint32	checkSum	CheckSum for this table.
Offset32	offset	Offset from beginning of TrueType font file.
uint32	length	Length of this table.

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

Table tags are the names given to tables in the OpenType font file. For requirements of Tag values, see Data Types, above.

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

Calculating Checksums

Table checksums are the unsigned sum of the uint32 units of a given table. In C, the following function can be used to determine a checksum:

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

> *Note:* This function implies that the length of a table must be a multiple of four bytes. In fact, a font is not considered structurally well-formed 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:

1. Set the checkSumAdjustment to 0.
2. Calculate the checksum for all the tables including the 'head' table and enter that value into the table directory.
3. Calculate the checksum for the entire font.
4. Subtract that value from 0xB1B0AFBA.
5. 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.

Font Collections

An OpenType Font Collection (formerly known as TrueType Collection) is a means of delivering multiple OpenType font resources in a single file structure. The format for font collections allows font tables that are identical between two or more fonts to be shared. Font collections containing outline glyph data (TrueType, CFF, CFF2 or SVG) are most useful when the fonts to be delivered together share many glyphs in common. By allowing multiple fonts to share glyph sets and other common font tables, font collections can result in a significant saving of file space.

For example, a group of Japanese fonts may each have their own designs for the kana glyphs, but share identical designs for the kanji. With ordinary OpenType 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 a Font Collection (as part of the TrueType specification) was intended to be used with fonts containing TrueType outlines, and this constraint was maintained in earlier OpenType versions, this is no longer a constraint in OpenType. Font collection files may contain various types of outlines (or a mix of them), regardless of whether or not fonts have layout tables present.

> *Note:* A variable font using OpenType Font Variations mechanisms is functionally equivalent to multiple non-variable fonts. Variable fonts do not need to be contained within a collection file. A collection file can include one or even multiple variable fonts, however, and may even combine variable and non-variable fonts.

The Font Collection File Structure

A font collection file consists of a single TTC Header table, one or more Offset Tables with Table Directories (each corresponding to a different font resource), and a number of OpenType 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 resource. A TTC file Table Directory has exactly the same format as a TTF file Table Directory. The table offsets in all Table Directories within a TTC file are measured from the beginning of the TTC file.

Each OpenType table in a TTC file is referenced through the Offset Table and Table Directory of each font which uses that table. Some of the OpenType 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.

A tool is available from Microsoft to help build .TTC files. The process involves paying close attention the issue of glyph renumbering in a font and the side effects that can result, in the 'cmap' table and elsewhere. The fonts to be merged must also have compatible TrueType instructions-that is, their preprograms, function definitions, and control values must not conflict.

Collection files containing TrueType glyph outlnes should use the filename suffix .TTC. Collection files containing CFF or CFF2 outlines should use the file extension .OTC.

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 OpenType fonts in the file, and an array of offsets to each Offset Table.

TTC Header Version 1.0:

Type	Name	Description
TAG	ttcTag	Font Collection ID string: 'ttcf' (used for fonts with CFF or CFF2 outlines as well as TrueType outlines)
uint16	majorVersion	Major version of the TTC Header, = 1.
uint16	minorVersion	Minor version of the TTC Header, = 0.
uint32	numFonts	Number of fonts in TTC
Offset32	offsetTable[numFonts]	Array of offsets to the OffsetTable for each font from the beginning of the file

TTC Header Version 2.0:

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

Font Tables

The TrueType rasterizer has a much easier time traversing tables if they are padded so that each table begins on a 4-byte boundary. Also, the algorithm for calculating table checksums assumes that tables are 32-bit aligned. For this reason, all tables must be 32-bit aligned and padded with zeroes.

Required Tables

Whether TrueType or CFF outlines are used in an OpenType 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

Tables Related to TrueType Outlines

For OpenType fonts based on TrueType outlines, the following tables are used:

Tag	Name
'cvt '	Control Value Table (optional table)
'fpgm'	Font program (optional table)
'glyf'	Glyph data
'loca'	Index to location
'prep'	CVT Program (optional table)
'gasp'	Grid-fitting/Scan-conversion (optional table)

Tables Related to CFF Outlines

For OpenType fonts based on CFF outlines, the following tables are used:

Tag	Name
'CFF '	Compact Font Format 1.0
CFF2	Compact Font Format 2.0
VORG	Vertical Origin (optional table)

It is strongly recommended that CFF OpenType fonts that are used for vertical writing include a Vertical Origin (VORG) table.

Multiple Master support in OpenType has been discontinued as of version 1.3 of the specification. The MMSD and MMFX tables that were defined in versions prior to version 1.3 are no longer supported.

Table Related to SVG Outlines

Tag	Name

'SVG '	The SVG (Scalable Vector Graphics) table

Tables Related to Bitmap Glyphs

Tag	Name
EBDT	Embedded bitmap data
EBLC	Embedded bitmap location data
EBSC	Embedded bitmap scaling data
CBDT	Color bitmap data
CBLC	Color bitmap location data
'sbix'	Standard bitmap graphics

OpenType fonts may also contain bitmaps of glyphs, in addition to outlines. Hand-tuned bitmaps are especially useful in OpenType fonts for representing complex glyphs at very small sizes. If a bitmap for a particular size is provided in a font, it will be used by the system instead of the outline when rendering the glyph.

Advanced Typographic Tables

Several optional tables support advanced typographic functions:

Tag	Name
BASE	Baseline data
GDEF	Glyph definition data
GPOS	Glyph positioning data
GSUB	Glyph substitution data
JSTF	Justification data
MATH	Math layout data

For information on common table formats, please see OpenType Layout Common Table Formats .

Tables used for OpenType Font Variations

Tag	Name
'avar'	Axis variations
'cvar'	CVT variations (TrueType outlines only)
'fvar'	Font variations
'gvar'	Glyph variations (TrueType outlines only)

HVAR	Horizontal metrics variations
MVAR	Metrics variations
STAT	Style attributes (required for variable fonts, optional for non-variable fonts)
VVAR	Vertical metrics variations

For an overview of OpenType Font Variations and the specification of the interpolation algorithm used for variations, see OpenType Font Variations Overview. For details regarding which tables are required or optional in variable fonts, see Variation Data Tables and Miscellaneous Requirements in the Overview chapter.

For information on common table formats used for variations, see OpenType Font Variations Common Table Formats.

Note that some variation-related formats may be used in tables other than the variations-specific tables listed above. In particular, the GDEF or BASE tables in a variable font can include variation data using common table formats. A CFF2 table in a variable font can also include variation data, though using formats that are specific to the CFF2 table.

Tables Related to Color Fonts

Tag	Name
COLR	Color table
CPAL	Color palette table
CBDT	Color bitmap data
CBLC	Color bitmap location data
'sbix'	Standard bitmap graphics
'SVG '	The SVG (Scalable Vector Graphics) table

Note that several of these tables were also listed in other sections for tables related to SVG outlines, and for tables related to bitmap glyphs.

Other OpenType Tables

Tag	Name
DSIG	Digital signature
'hdmx'	Horizontal device metrics
'kern'	Kerning
LTSH	Linear threshold data
MERG	Merge

'meta'	Metadata
STAT	Style attributes
PCLT	PCL 5 data
VDMX	Vertical device metrics
'vhea'	Vertical Metrics header
'vmtx'	Vertical Metrics

Note that the STAT table is required in variable fonts. Also, the 'hdmx' and VDMX tables are not used in variable fonts.