MakeOTF v2.5 OpenType/CFF compiler
User Guide

Overview

MakeOTF is a tool designed to create an OpenType® font from a source font file and from a text file containing high-level descriptions of OpenType layout features. It is designed to run as command-line tool: a command typed into the Terminal window on Mac OS® X, or in the DOS window on Windows®. Note that MakeOTF is only a compiler of font data, not a font editor.

MakeOTF requires a number of source files that can all be specified through options for the `makeotf` command:

font. Usually named `font.pfa` or `cidfont.ps`. This can be either a Type 1 or CID font file, a TrueType font file, or an OpenType/CFF font file. Note that only the glyph outlines are taken from the source font..

features. A text file which provides the definitions for the OpenType layout feature rules, and specifies values for some of the OpenType table fields that will override the default values calculated by MakeOTF.

FontMenuNameDB. A text file which provides the Windows and Mac menu names for the font.

GlyphOrderAndAliasDB. This file serves three purposes. One purpose is to establish the glyph ID order in the font. Another is to allow glyphs in the output `.otf` file to have different names than the glyphs in the input source font data. This permits developers to use friendlier names during development of a font, and use different names in the final OpenType file. The third is to provide Unicode® values for the glyphs. MakeOTF will provide Unicode values by default for some glyphs, but not all.

An additional file, **fontinfo**, is not required, but if present, will be examined for keywords that cause MakeOTF to set some specific options.

In order to avoid having to enter the options to specify the necessary files every time you want to build an OpenType font, MakeOTF can save a font project file that stores all desired options. MakeOTF will always write the last set of options used, to a file named `current.fpr` located in the same directory as the input font file. (Project files can also be saved under other names)

Options can be added to the `makeotf` command to set parameters that change how MakeOTF builds an OpenType font. When options conflict, the last one on the command line will override any earlier conflicting options.

Using MakeOTF

MakeOTF comes in two parts which can actually be called independently:

`makeotfexe` is a program written in C, and is actually the tool that builds the OpenType font file. It requires, however, that all the source files be explicitly specified with options on the command line.

`makeotf` is a commadn shell that calls the Python™ script MakeOTF.py. This will look for the source files in some standard locations, and fill in default values for options. It can also read the options needed from a project file (`current.fpr`), which means that for a particular font, one only needs to type-in all the options once. When MakeOTF.py has gathered all the source files, it will call the `makeotfexe` program.

In general, one should invoke MakeOTF.py with the `makeotf` command. This way, the last set of options used will always be recorded in a project file.

The first step in using MakeOTF is to assemble the source files. It is often a good idea to organize the files in a directory tree, like this:

```
MyFont Family/
    FontMenuNameDB
    GlyphOrderAndAliasDB
    Roman/
        features.family
        MyFont-Bold/
            font.pfa
            features
            features.kern
        MyFont-Regular/
            font.pfa
            features
            features.kern
    Italic/
        features.family
        MyFont-Italic/
            font.pfa
            features
            features.kern
        MyFont-BoldItalic/
            font.pfa
            features
            features.kern
```

The idea is that some data, such as the files `FontMenuNameDB` and `GlyphOrderAndAliasDB`, will be shared between all members of the type family, whereas other data will only apply to a subgroup of the family. By placing the shareable data at a higher level in the directory tree, one can avoid having multiple copies of the same data, which means there will be less files to edit, in case the data needs to be changed. This is most important for the `features.family` file.

Usually, positioning rules such as kerning (`features.kern`) are specific to each font, but most (or all) of the substitution rules are likely to be the same for all fonts. A good practice is to have a `features` file for each family member, in the same directory as the source font file, and then use an `include` statement to reference feature files located higher in the directory tree. In the example above, there are two separate `features.family` files, one for Roman and another for Italic. These can, in turn, be shared between all family members under each branch. The need for two different `features.`

`family` files, comes from the fact that Roman and Italic styles might not have the same number of substitution (GSUB) rules — Italic tends to have more.

Once one has assembled all the necessary files, the next step is to open a command window — the Terminal program on Mac OS X, or the Command program on Windows —, and use the `cd` command to set the current working directory to the directory which contains the set of source files to compile the OpenType font:

```
cd <path to font directory>
```

and then type the `makeotf` command, with the options needed. For example:

```
makeotf -f myfont.pfa -ff myfeatures -b -r
```

or

```
makeotf -fp myproject.fpr
```

MakeOTF options

Option	Setting	Description
`-fp`	`<file path>`	Specify path to project file. If no path is given, the default is `current.fpr`. MakeOTF will read the options from this project file. The project file contains values only when they differ from the default value. The `-fp` option can be used with other options, but must always be first. Additional options will override those read from the project file. For example, `-fp release.fpr -o test.otf` will build an OpenType font with all the options in the `release.fpr` project file, but will write the output file as `test.otf`, instead of `<PostScript-Name>.otf`.
`-f`	`<file path>`	Specify path to input font. If not provided, MakeOTF assumes the file name is `font.pfa`.
`-o`	`<file path>`	Specify path to output font. If not provided, MakeOTF assumes the file name is `<PostScript-Name>.otf`.
`-b`		Set style to Bold. Affects style-linking. If not provided, MakeOTF assumes that the font is not bold.
`-i`		Set style to Italic. Affects style-linking. If not provided, MakeOTF assumes that the font is not italic.
`-ff`	`<file path>`	Specify path to features file. If not provided, MakeOTF assumes the file name is `features`.
`-gs`		Omit any glyphs that are not named in the GOADB file. Works only if either the -ga or -r options is specified.
`-mf`	`<file path>`	Specify path to FontMenuNameDB file. If not provided, MakeOTF will look in the current directory for a file named `FontMenuNameDB`, and then one directory up, and finally two directories up.

Option	Setting	Description
`-gf`	`<file path>`	Specify path to GlyphOrderAndAliasDB file. If not provided, MakeOTF will look in the current directory for a file named `GlyphOrderAndAliasDB`, and then one directory up, and finally two directories up. Also, if this option is specified, and the `-r` or `-ga` options are NOT specified, the effect is to use the Unicode assignments from the third column of the GOADB without renaming the glyphs.
`-r`		Set release mode. This option turns on subroutinization, applies the GlyphOrderAndAliasDB file, and removes the word *Development* from the name table Version (Name ID 5) string.
`-S`		Turn on subroutinization.
`-ga`		Apply the GlyphOrderAndAliasDB file. Use when the `-r` option is NOT specified.
`-rev`	`[<number>]`	Attempts to edit the feature file before makeotfexe is run, in order to increment the head table fontRevision field. This only works if the head table override is already defined in the features file. Without the optional version number, increments the version number by 5. With an integer argument, it increments the minor version by that number. With a fractional argument, it sets the version to the fractional argument; the number must then be decimal with three decimal places, e.g. "1.045", not '1.45'.
`-osbOn`	`<number>`	Turn on the specified bit number(s) in the OS/2 table fsSelection field. In order to turn on more than one bit, must be used more than once. `-osbOn 7 -osbOn 8` will turn on bits 7 and 8. See section below on *New OS/2 Bits.*
`-osbOff`	`<number>`	Turn off the specified bit number(s) in the OS/2 table fsSelection field. Can be used more than once to turn OFF more than one bit at a time. `-osbOff 7 -osbOff 8` will turn off bits 7 and 8. See section below on *New OS/2 Bits.*
`-osv`	`<number>`	Set version number of the OS/2 table. The default value is 3 if none of the bits specified only in version 4 and later are used; otherwise, the default version is 4. See section below on *New OS/2 Bits.*
`-addn`		Replace the *.notdef* glyph in the source data (if any) with a standard *.notdef* glyph, that will match the font's weight and width.
`-adds`		Create any Apple Mac Symbol glyphs missing from the font. Added glyphs will match the font's weight and width. .
`-serif`		Specify that any added glyph will use the serif Multiple Master built-in glyph data.
`-sans`		Specify that any added glyphs will use the sans-serif Multiple Master built-in glyph data.
`-cs`		Override the heuristics, and specify the script ID for the Mac cmap subtable.

Option	Setting	Description
`-cl`		Override the heuristics, and specify the language ID for the Mac cmap subtable.
`-cm`	`<file path>`	Specify path to CID CMap encoding file for the font Mac encoding. (CID-keyed fonts only)
`-ch`	`<file path>`	Specify path to CID CMap encoding file for the font Unicode UTF-32 encoding for horizontal glyphs. (CID-keyed fonts only)
`-cv`	`<file path>`	Specify path to CID CMap encoding file for the font Unicode UTF-32 encoding for vertical glyphs. (CID-keyed fonts only)
`-ci`	`<file path>`	Specify path to Unicode Variation Sequence specification file.
`-dbl`		Map glyph names to two Unicode values rather than one. This was the default behaviour of `makeotf` in FDK 1.6 and earlier. The Adobe Type Department now discourages this practice. The option exists only to allow building fonts that match original versions. See `makeotf -h` for the hard-coded list of glyphs.
`-dcs`		Set OS/2.DefaultChar to the Unicode value for 'space', rather than '.notdef. The latter is correct by the OT spec, but QuarkXPress 6.5 requires the former in order to print OTF/CFF fonts.
`-fi`	`<file path>`	Path to the fontinfo file. If no path is given, the default is to look for first 'fontinfo', then 'cidfontinfo', in the current directory. Used to set some default values. This are overridden by any conflicting settings in the project file and then by command-line options. This option is processed before any others, so if the path is relative, it is relative to the current working directory. All other relative paths are relative so the source font's parent directory..
`-sp`	`<file path>`	Save the current options to the file path provided, as well as to the `current.fpr` file.
`-nb`		Turn off the Bold style. Can be used to override a project file setting, otherwise has no effect.
`-ni`		Turn off the Italic style. Can be used to override a project file setting, otherwise has no effect.
`-nS`		Turn off subroutinization.
`-nga`		Turn off applying the GlyphOrderAndAliasDB file.
`-naddn`		Turn off adding a standard *.notdef*. Can be used to override a project file setting, otherwise has no effect.
`-nadds`		Turn off adding Apple symbol glyphs. Can be used to override a project file setting, otherwise has no effect.

Options are applied in the order in which they are specified: `-r -nS` will not subroutinize a font, but `-nS -r` will subroutinize a font. Option values are read (in order of increasing priority) from first the **fontinfo file** keyword/value pairs, then the specified **project file**, if any, and then from the **command line**, in order from left to right.

Subroutinization is a process by which common elements in a set of glyphs are decomposed in separate subroutines. This can reduce the size of the final font but can require extreme amounts of memory and time for a large font, such as CID fonts. MakeOTF may need as much as 64 Mbytes of memory for large Roman fonts, and will do most with only 32 Mbytes, but it may need 768 Mbytes of memory to subroutinize a 5-Mbyte CID font. Subroutinizing is reasonably fast when done all in memory: a few minutes for a Roman font, half an hour to three hours for a CID font. However, if the system must use virtual memory, the time required can increase by a factor of 20 or more. Subroutinizing is likely to be useful for Roman fonts, and for Korean CID fonts. Japanese and Chinese CID fonts usually only yield a few percent size reduction with subroutinizing, due to fewer repeating path elements.

FontMenuNameDB - Version 2.

Previous versions of MakeOTF used a different version of the FontMenuNameDB file, and wrote the Macintosh font menu names differently than the Windows font menu names, and not according to the OpenType spec. This is because of some history of the early efforts to get OpenType fonts working on the Mac OS. However, for some years Apple has been following the OpenType spec when making Apple OpenType fonts, and has fully supported the OpenType font menu names. As a result, this version of MakeOTF has implemented new syntax for the FontMenuNameDB, and will create the name table according to the OpenType spec when this new syntax is used.

Fonts made with earlier versions of MakeOTF will not be disadvantaged, as all Adobe fonts to date and many third-party fonts were made this way, and all programs look first to the Windows font menu names when they exist, as this is where the style-linking names can most reliably be found. The earlier version of the FontMenuNameDB may still be used. The main reason to change is that the newer version is easier to explain and understand.

The FontMenuNameDB file is used to specify font menu-naming information. It consists of a text file that may contain information for several different fonts. Each font entry starts with the `<PostScript Name>`, and is followed by the various menu names. These names are used to build the name table strings that describe the font. This allows one to put the menu names for all the fonts in a single file, making it easier to coordinate menu names across a family.

Font Entry Description	
`[<PostScript Name>]`	
`f=`	`Preferred Family name`
`s=`	`Subfamily name` (a.k.a. `Style name`)
`l=`	`compatible family menu name`
`m=1,`	`Macintosh compatible Full name`

Each `FontMenuNameDB` entry is composed of at least:

`Preferred Family name`, which is specified with the key `f=`, followed by the family name;

In many programs' font menus, the fonts will be listed in a cascading menu showing the Family names in the first level, followed by a pop-out list of Style names, containing all fonts that share the same Family name. In other programs, the menus will show a "flat" list of all fonts using the *Full name*. `makeotf` will build this by concatenating the `Family name`, a space character, and the `Style name`.

However, additional names may need to be supplied. This happens because style-linking (selecting another font of the same family by applying Bold or Italic options to the current font) only allows for families to have up to four members. This means that only four fonts can share the same Family name. If the family has more than four members, or has members which are not style-linked, it is then necessary to divide the family into sub-families which are style-linked, and to provide each sub-family with a unique `compatible` family `menu name`.

Only fonts that share the same compatible family menu name can be style-linked, and each 4-element group can only contain one font of each of the styles *Regular, Bold, Italic,* and *BoldItalic*.

The `compatible family menu name` is specified with the key `l=`, followed by the name. Whenever this name needs to be defined, a compatible *Full name* for the Mac must also be specified. The `Macintosh compatible menu name` is specified with the key `m=1,`, followed by the name. This name is normally built by concatenating the compatible family menu name, a space character, and the appropriate choice of one of the four supported styles.

Although these naming conventions are necessary for correct style-linking, they are not sufficient. The font style flags for Bold or Italic must also be set correctly. This can be done either with the `makeotf` options `-b` and `-i`, or by providing a `fontinfo` file located in the same directory as the font and which contains the key/value pairs `IsBoldStyle true` and `IsItalicStyle true`, with `true` or `false` being set appropriately.

Compatible family menu names are also used in font menus by applications that are not OpenType savvy.

If the family only has four faces that are meant to be style-linked, then the compatible and preferred family menu names are the same, and only the "f=" entry is needed. The names specified with the key `f=` will then be written to the name table Name ID 1. If there is a compatible family name entry which differs from the entry supplied with the key "f=" , then the "`f=" family` name is written to Name ID 16. The compatible name from `l=` is then written to name table Name ID 1, and the name table Name ID 2 is chosen by `makeotf` to be one of *Regular, Bold, Italic,* or *BoldItalic,* based on the font's style flag. The value set by `m=1,` is written to the Mac platform name table Name ID 18.

The key "s=" only needs to be used when you want a style name which is different than the choice from the styles *Regular, Bold, Italic,* and *BoldItalic* that would be dictated by the font's style. When used, and different than the default style name, the style name is written to name table NameID 17. Otherwise, it is written to name table NameID 2.

For complete description of these issues, please read the OpenType specification section on the name table, available from `http://partners.adobe.com/public/developer/opentype/index_name.html`.

Examples

Regular font of Adobe® Garamond® Pro

```
[AGaramondPro-Regular]
f=Adobe Garamond Pro
s=Regular
```

The `l=` key is not used, so the compatible family menu name is the same as the Preferred Family name. The `m=1,` key is not used, so the Macintosh compatible menu name built by MakeOTF will be *Adobe Garamond Pro Regular* . The key "s=" is not actually necessary.

Bold font of Adobe Garamond Pro

```
[AGaramondPro-Bold]
f=Adobe Garamond Pro
s=Bold
```

The `l=` key is not used, so the compatible family menu name is the same as the Preferred Family name. This font will be style-linked with the *Regular,* because they share the same compatible family menu name.

Italic font of Adobe Garamond Pro

```
[AGaramondPro-Italic]
f=Adobe Garamond Pro
s=Italic
```

Same as above.

Semibold font of Adobe Garamond Pro

```
[AGaramondPro-Semibold]
f=Adobe Garamond Pro
s=Semibold
l=Adobe Garamond Pro Sb
m=1,Adobe Garamond Pro Sb
```

This font needs to be part of a new style-linking subgroup. This means the key `l=` is used to set a `compatible family menu name` different from Preferred `Family name`. In consequence, a `Macintosh compatible Full name` is also assigned with the key `m=1,`. The default style will be *Regular.* The compatible family menu names are abbreviated to *Adobe Garamond Pro Sb* rather than *Adobe Garamond Pro Semibold,* in order to keep the menu names, of the remaining fonts belonging to this style-linked subgroup, within the 31-character limit. The key "s=" had to be used, as the preferred style name is not "Regular", the style-linking style.

Semibold Italic font of Adobe Garamond Pro

```
[AGaramondPro-SemiboldItalic]
f=Adobe Garamond Pro
s=Semibold Italic
l=Adobe Garamond Pro Sb
m=1,Adobe Garamond Pro Sb Italic
```

This font is part of the same style-linking subgroup as the font in the previous example. In order to be style-linked, they share the same `l=` key, but the default style will be set to *Italic* in this case. The `Macintosh compatible Full menu name` was be built by appending the default style to the `compatible family menu name`.

Note on length restrictions

Because of limitations in operating systems and common applications, it is recommended that none of these keys contain names longer than 31 characters. This results in menu names for legacy environments being 31 characters or less.

The OpenType menu name (used in Adobe InDesign® and future Adobe applications) may be thought of as the combination of the `f=` and `s=` keys. Since each of these can be up to 31 characters, the OpenType menu name can be up to 62 characters.

Note on accented/extended characters

To use accented characters in Latin menu names, one needs two `f=` entries, one for Windows and one for Mac. The Windows entry specifies the character by Unicode. The Mac entry specifies it by the hex value of the Mac char code from the font's mac cmap table, when different from the ASCII value.

Arnold Böcklin	
`f=Arnold B\00f6cklin`	Windows
`f=1,Arnold B\9acklin`	Mac

`00f6` is the Unicode for *odieresis* (ö). Unicode *hex* values must have 4 digits.

`9a` is MacRoman encoding for *odieresis*. Mac char code hex values must have 2 digits for Western script encodings, and may have 2 or 4 digits for CJK encodings.

In general, the `f=`, `s=`, and `l=` can be extended to set non-Latin strings by adding the triplet (platform code, script code, language code) after the equal sign (`=`). The values are the same as described for the name table name ID strings. For example:

小塚明朝 Pro (Kozuka Mincho® Pro)	
`[KozMinPro-Bold]`	
`f=3,1,0x411,\5c0f\585a\660e\671d Pro`	Windows, Unicode, Japanese
`f=1,1,11,\8f\ac\92\cb\96\be\92\a9 Pro`	Mac, Japanese, Japanese

The AFDKO contains the file **FDK/Tools/SharedData/**`FontMenuNameDB`, which shows the entries for several families of the Adobe Type Library.

FontMenuNameDB - Version 1.

Versions of MakeOTF prior to FDK 2.5 used a similar synax in the FontMenuNameDB file. When this older version syntax is used, MakeOTF will build the name table font menu names as it did in FDK version 2.0 and earlier. These earlier versions built the Windows platform names the same. however, only the Preferred Family and Style names were written in the Mac platform name strings, and in respectively name ID 1 and name ID 2. The key "c=" set the compatbile family name for the Windows platform. There was no way to specify a compatible family name for the Mac platform. The key "c=1," set instead a compatible value for the Mac platform Full Name String, name ID 18.

.

Font Entry Description	
[<PostScript Name>]	
f=	Family name
s=	Subfamily name (a.k.a. Style name)
c=	Windows compatible menu name
c=1,	Macintosh compatible menu name

If the key "c=" is used. then MakeOTF will build the older style name table. If the keys "l=" or "m=" are present, it will build the newer style name table . If none of these are present, then there is no difference in how the name table is built.

GlyphOrderAndAliasDB (GOADB)

The GOADB file is used to rename and to establish an order for the glyphs in a font. It is a simple text file with one line per glyph name. Each line contains at least two fields, and optionally three fields. The first field is the final glyph name to be used in the output font. The second field is the 'friendly' name used in the source font data. The third field is a Unicode value, specified in the form uniXXXX or uXXXX, where XXXX is a Unicode value. The source font is not required to have any glyphs that are named in the `GlyphOrderAndAliasDB` file. There is a sample version of this file at the folder **FDK/Tools/SharedData/**.

It should be noted that the ordering, renaming, and Unicode override operations are applied only if the `-r` option or the `-ga` option is specified. These operations work as follows.

1) **Establish a specific glyph order in the output OpenType font.** Any glyph names in *Appendix A – Standard Strings* of Adobe's Technical Note #5176, *The Compact Font Format Specification*, will be ordered first in the font, in the order given in the CFF specification. All other glyphs will be ordered as in the GOADB file. Glyphs not named in either the CFF specification nor in the GOADB file, will be ordered as they were in the original source font. The CFF specification can be found at `http://partners.adobe.com/public/developer/en/font/5176.CFF.pdf`

2) **Rename glyphs in the source font to different names in the output OpenType font.** Note that both the source font file and the features definition file must use the same glyph names – one cannot use a source font file with development names, together with a features file that contains final names, unless the options `-r` or `-ga` are used.

3) **Override the default Unicode encoding by MakeOTF.** MakeOTF will assign Unicode values to glyphs in a non-CID font when possible. (For a CID font, the Unicode values are provided by the Adobe CMap files.) The rules used are as follows:

   a) If the third field of the GOADB record for a glyph contains a Unicode value in the form uniXXXX or uXXXX (where XXXX stands for a Unicode value), assign that Unicode value to the glyph. Else b);

   b) If a glyph name is in the *Adobe Glyph List For New Fonts v1.6* (**FDK/Technical Documentation/ GlyphNames/**`aglfn13.txt`), use the assigned Unicode value. Else c);

c) If the glyph name is in the form uniXXXX or uXXXX (where XXXX stands for a Unicode value), assign the Unicode value. Else d);

d) Do not assign any Unicode value.

Note that MakeOTF cannot re-order glyphs when the source font is a TrueType or OpenType/TTF font: the glyph order in the source font and the glyph order in the GlyphOrderAndAliasDB file must be the same. It can, however, still rename glyphs and assign Unicode values.

Note that MakeOTF no longer assigns glyphs Unicode values from the Private Use Area (PUA) block. If such Unicode values are needed, they must be specified in a GOADB file.

fontinfo

The fontinfo file is a simple text file containing key-value pairs. Each line contains two white-space separated fields. The first field is a keyword, and the second field is the value. `makeotf` will look for a **fontinfo** file in the same directory as the source **font** file, and, if found, use it to set some default values. These values will be overridden if they are also set by a project file, and then by any `makeotf` command-line options. The keywords and values currently supported are:

Keyword	Values	Effect
`IsBoldStyle`	`true/false`	Set the font style to Bold. Same as the command-line option `-b`
`IsItalicStyle`	`true/false`	Set the font style to Italic. Same as the command-line option `-i`
`PreferOS/2TypoMetrics`	`true/false`	Set the OS/2 table fsSelection bit 7, USE_TYPO_METRICS. Same as the command-line option `-osbOn 7`.
`IsOS/2WidthWeigthSlopeOnly`	`true/false`	Set the OS/2 table fsSelection bit 8, WEIGHT_WIDTH_SLOPE_ONLY. Same as the command-line option `-osbOn 8`.
`IsOS/2OBLIQUE`	`true/false`	Set the OS/2 table fsSelection bit 9, OBLIQUE. Same as the command-line option `-osbOn 9`.

Adobe character code map (CMap)

A CMap file maps character codes to glyph selectors. It is only necessary for CID fonts. This file may be located anywhere within the file system, as long as the directory path is stored in the `FontEnvironment.txt` file, or the file is chosen explicitly via the UI. The default CMap files are automatically selected by MakeOTF when the `cidfontinfo` file is selected.

The Mac CMap file provides the encoding for a Mac platform cmap subtable in the OpenType font cmap table. This is required. The script and language of this Mac platform cmap subtable is determined by heuristics in MakeOTF, but can be specified by MakeOTF options. These apply only to CID source fonts.

The Vertical and Horizontal CMap files supply the Unicode encoding for all the glyphs in the font. Note that for CID fonts, only glyphs named in these three files are encoded in any of the OpenType font cmap table subtables. The many alternates can often be accessed only through OpenType layout features.

When looking for default CID CMap files, MakeOTF uses the following rules:

1) The default CMap files will be in a subdirectory of the default path which is specified in the `FontEnvironment.txt` file. If this is a relative rather than absolute path, it will be assumed to be under **FDK/Tools/SharedData/**.

2) The CMap files will be under a directory which is named after the Registry-Order-Supplement from the `cidfontinfo` file, e.g. Adobe-Japan1-4, Adobe-GB1-4.

3) The files will be named:

for R-O	Mac CMap	Unicode H CMap	Unicode V CMap
Adobe-Japan1:	83pv-RKSJ-H	UniJIS-UTF32-H	UniJIS-UTF32-V
Adobe-Japan2:	83pv-RKSJ-H	UniJIS-UTF32-H	UniJIS-UTF32-V
Adobe-GB1:	GBpc-EUC-H	UniGB-UTF32-H	UniGB-UTF32-V
Adobe-CNS1:	B5pc-H	UniCNS-UTF32-H	UniCNS-UTF32-V
Adobe-Korea1:	KSCpc-EUC-H	UniKS-UTF32-H	UniKS-UTF32-V

If one wants to use a CMap, which would not be found by these assumptions, one must specify them explicitly on the command-line while using `makeotf`.

For further information on CMaps, please read the many Adobe's Technical Notes available from `http://partners.adobe.com/public/developer/font/index.html#ckf`.

Unicode Variation Sequence (UVS) File

A UVS file is a list of records, each of which specifies the glyph which should be displayed for a given combination of a Unicode encoding value and a Unicode Variation Selector. You can read about Unicode Variation Sequence in the publication "Unicode Technical Stadard #37 , Ideographic Variation Database , at http://unicode.org/reports/tr37/. The format depends on the source font format: CID-keyed fonts require three fields in each record, non-CID fonts require only two.

For CID-keyed fonts:

The makeotf command will look for a UVS file under **FDK/Tools/SharedData/Adobe CMAPS/<R-O-S>,** where <R-O-S> stands for the font's CID Registry-Order-Supplement. The file will be assumed to be named "<R-O>_sequences.txt . For a CID font with the R-O-S "Adobe-Japan1-6". the default UVS file path woudl be "**FDK/Tools/SharedData/Adobe CMAPS/Adobe-Japan1-6/Adobe-Japan1_sequences.txt**" . The option "-ci <path>" may be used to specify a UVS file path other than the default

The file format is a n ASCII text file. Each line represents one Unicode Variation Sequence(UVS). A line may be blank, or contain a comment that starts with the charater "#".

Each line contains three fields. Fields are separated by a semicolon followed by whitespace. The fields are:

1) Variation Sequence. Two hexadecimal value, with no leading “x” or “0x, separated by a space. The first is the Base Unicode Value, and the second is the Variation Selector value..
2) Adobe CID Registry and Order. The CID Registry and Order names, joined by a hyphen.
3) CID Value. A decimal value, prefixed by “CID+”.

Example UVS file:

```
# Registered Adobe-Japan1 sequences
# Version 07/30/2007
# Prepared by Ken Lunde (lunde@adobe.com), Adobe Systems Incorporated
3402 E0100; Adobe-Japan1; CID+13698
3402 E0101; Adobe-Japan1; CID+13697
3402 E0102; Adobe-Japan1; CID+13699
3405 E0100; Adobe-Japan1; CID+15387
....
```

For non CID-keyed fonts:

The path to the variation sequence file must be specified with the option ‘-ci <file path>’.
The format for the UV and UVS sequence is the same. However, the Registry-Order field is omitted, and the glyph is identified by a glyph name from the GOADB file.

Example UVS file for a nonCID-keyed font:

```
3402 E0100;  checkbox
3402 E0101;  checkedbox
```

New OS/2 Bits

In 2006, Microsoft and Adobe began discussions to define three new bit values in the OS/2 table fsSelection field: bit 7 USE_TYPO_METRICS, bit 8 WEIGHT_WIDTH_SLOPE_ONLY, and bit 9, OBLIQUE. These bits have meaning only in OS/2 table version 4 and later; in earlier versions, they are reserved, and should be set off. The Adobe Type Department will set the bits in our new fonts.

USE_TYPO_METRICS. When bit 7 is on, programs are supposed to use the OS/2 table sTypoAscent/Descent/LineGap for vertical line spacing. This bit was defined because the Windows layout libraries have been using the OS/2 table winAscent/Descent values instead. The next release of Windows Presentation Foundation, on which new versions of Microsoft programs such as Word will be based, and which is due in late 2006, will use the sTypo values instead – but only if this bit is on. This bit certifies that the OS/2 sTypo values are good, and have the side effect of being present only in new fonts, so that reflow of documents will happen less often than if Microsoft just changed the behaviour for all fonts. For example, it is rumored that some TrueType fonts have bad values for the sTypo values, making it undesirable to apply the sTypo values for all existing fonts.

The next two bits have to do with the fact that future generations of Microsoft programs, and programs from other vendors that use the Windows Presentation Foundation (WPF) libraries, will use the Cascading Style Sheet (CSS) v.2 specification for specifying fonts. The provisions of this specification can be seen at `http://www.w3.org/TR/CSS2/fonts.html`.

In brief, a CSS font declaration within a document will describe a font by specifying the family name, and one or more properties; there is no way to refer directly to a specific font. The possible styles are *Regular, Italic* and *Oblique.* Weight and width are specified with separate keywords, with a limited and defined set of possible values. An example (in a simplified syntax) is "**Family:** *TimesStd*, **Style:** *Italic*, **Weight:** *700*, **Width:** *Normal*". One requirement of a CSS family is that each font has to have a different set of properties. There cannot be two fonts in a CSS family which both have the properties "**Style:** *Italic*, **Weight:** *700*, **Width:** *Normal*". For OpenType fonts, the name table Preferred Family Name is used as the CSS family name. Nonetheless, OpenType font families may well contain more than one font with a particular set of properties, but this fact will create a conflict in a CSS-based environment. When WPF encounters one of such families, it will divide the family into groups, so that only one font in each group has a given set of properties. However, that will allow WPF to provide new family names for each group, and these might not correspond to the type designer's initial intentions.

Similarly, WPF will infer the font's properties by analyzing its Preferred Family and Style names, and may mistakenly conclude that any two fonts in a family cannot be uniquely assigned different sets of font properties. In this case, WPF will also divide the family into groups, and generate new family names for them. The WEIGHT_WIDTH_SLOPE_ONLY bit was added to deal with this latter case. When this bit is set in a font's OS/2 table fsSelection field, the application is requested to trust that all fonts sharing the same Preferred Family name, will differ from all the remaining with respect to their CSS properties. That given, the application can use the OpenType name table font menu names.

WEIGHT_WIDTH_SLOPE_ONLY. When bit 8 is on, a program can be confident that all the fonts which share the same Preferred Font Family Name will all differ only in weight, width, and slope. The Preferred Font Family Name is defined as name table Windows platform Name ID 16 Preferred Family Name, or, if name ID 16 is not present, then Windows platform Name ID 1 Family Name. If the OS/2 table version is 4 and this bit is not on, the program will use heuristics to divide the faces with the same Preferred Family Name into CSS family groups, and assign family and property names.

Yet another issue, is that WPF uses heuristics applied to the font menu names to determine if the font has the Oblique style. The Oblique style can be used for either a font that was designed as a slanted form of a typeface – as opposed to a true italic design –, or for a font which has been created by a program by algorithmically slanting a base font. There is no information in current OpenType fonts to indicate whether a font is Oblique. The only way to know if a current OpenType font is Oblique is through the use of heuristics based on analyzing the font name. However, no set of heuristics will be perfect. The Oblique bit was proposed in order to solve this problem, thus providing a way to clearly indicate if the font should be assigned the CSS Oblique style. As an example from the CSS specification, a Regular font within a family could be classified as Oblique just because the word "Inclined" is used in its name.

OBLIQUE. When bit 9 is on, programs that support the CSS2 font specification standards will consider the font to have the Oblique style, otherwise not. If the document specifies an Oblique style font, and no Oblique font is present within the CSS font family, then the application may synthetically create an Oblique style by slanting the base font. This will occur even if there is an Italic font within the same CSS font family. However, if a document font specification requires an Italic style, but only an Oblique font is available, then the latter will be used.

An additional proposal suggests that if this bit is not set and the OS/2 version is 4 or greater, then the application would look for Windows platform names ID 21 and 22. If present, name ID 21 would supply the CSS compatible family name, and name ID 22 would supply the CSS-compatible property name. As of the writing of this document, the status of this proposal is still uncertain – please check the latest OpenType specification. If this proposal is accepted, the additional names can be specified in the feature file.

Note that if any of these three new bits is turned on, with the option `-osb0n <n>`, then `makeotf` will require that all three values be explicitly set to be *on* or *off*. This is because, if any of new the bits is set *on*, `makeotf` will by default set the OS/2 version number to 4. When the version number is 4, then both the on and the off setting has a specific meaning for each bit.

Synthetic Glyphs

MakeOTF includes two Multiple Master fonts built-in, one serif and one sans-serif. With these it can synthesize glyphs that match (more or less) the width and weight of the source font. It requires the glyphs *zero* and *O* to be present in the font, in order to determine the required weight and width. If the option `-adds` is used, the list of glyphs to generate will be derived from the concatenation of the following three groups of glyphs:

1) Euro

2) Apple Symbol glyphs. These glyphs were formerly supplied by the Macintosh ATM™ and the Laserwriter® drivers. This is no longer true for OpenType fonts.

3) A miscellany of glyphs missing from some of the Adobe Type Library fonts, which were just a few glyphs short of containing the full Adobe PostScript Standard glyph set.

Synthetic Glyphs					
€	Euro	∆	Delta	Ω	Omega
≈	approxequal	^	asciicircum	~	asciitilde
@	at	\	backslash	|	bar
¦	brokenbar	¤	currency	†	dagger
‡	daggerdbl	°	degree	÷	divide
=	equal	℮	estimated	⁄	fraction
>	greater	≥	greaterequal	∞	infinity
∫	integral	<	less	≤	lessequal
ℓ	litre	¬	logicalnot	◊	lozenge
−	minus	×	multiply	≠	notequal
№	numbersign	½	onehalf	¼	onequarter
¶	paragraph	∂	partialdiff	‰	perthousand
π	pi	+	plus	±	plusminus

∏	product	"	quotedbl	'	quotesingle
√	radical	§	section	∑	summation
¾	threequarters	0	zero		

The glyphs are synthesized from the MM fonts which MakeOTF has built-in. It will try to match the glyph width of the *zero* and the dominant stem width of the target font. However, MakeOTF cannot stretch the MM font data to match very thick strokes, very wide glyphs, and it cannot match the design's stem contrast.