

NAME

Ttf2pt1 – A True Type to PostScript Type 1 Font Converter

SYNOPSIS

```
ttf2pt1 [-options] ttf font.ttf [Fontname]
```

or

```
ttf2pt1 [-options] ttf font.ttf -
```

DESCRIPTION

Ttf2pt1 is a font converter from the True Type format (and some other formats supported by the FreeType library as well) to the Adobe Type1 format.

The versions 3.0 and later got rather extensive post-processing algorithm that brings the converted fonts to the requirements of the Type1 standard, tries to correct the rounding errors introduced during conversions and some simple kinds of bugs that are typical for the public domain TTF fonts. It also generates the hints that enable much better rendering of fonts in small sizes that are typical for the computer displays. But everything has its price, and some of the optimizations may not work well for certain fonts. That's why the options were added to the converter, to control the performed optimizations.

OPTIONS

The first variant creates the file `Fontname.pfa` (or `Fontname.pfb` if the option '**-b**' was used) with the converted font and `Fontname.afm` with the font metrics, the second one prints the font or another file (if the option '**-G**' was used) on the standard output from where it can be immediately piped through some filter. If no `Fontname` is specified for the first variant, the name is generated from `ttffont` by replacing the `.ttf` filename suffix.

Most of the time no options are necessary (with a possible exception of '**-e**'). But if there are some troubles with the resulting font, they may be used to control the conversion. The **options** are:

- **-a** – Include all the glyphs from the source file into the converted file. If this option is not specified then only the glyphs that have been assigned some encoding are included, because the rest of glyphs would be inaccessible anyway and would only consume the disk space. But some applications are clever enough to change the encoding on the fly and thus use the other glyphs, in this case they could benefit from using this option. But there is a catch: the X11 library has rather low limit for the font size. Including more glyphs increases the file size and thus increases the chance of hitting this limit. See `app/X11/README` for the description of a patch to X11 which fixes this problem.
- **-b** – Encode the resulting font to produce a ready `.pfb` file.
- **-d suboptions** – Debugging options. The suboptions are:

a – Print out the absolute coordinates of dots in outlines. Such a font can not be used by any program (that's why this option is incompatible with '**-e**') but it has proven to be a valuable debugging information.

r – Do not reverse the direction of outlines. The TTF fonts have the standard direction of outlines opposite to the Type1 fonts. So they should be reversed during proper conversion. This option may be used for debugging or to handle a TTF font with wrong direction of outlines (possibly, converted in a broken way from a Type1 font). The first signs of the wrong direction are the letters like "P" or "B" without the unpainted "holes" inside.

- **-e** – Assemble the resulting font to produce a ready `.pfa` file.

[S.B.: Personally I don't think that this option is particularly useful. The same result may be achieved by piping the unassembled data through `tlasm`, the Type 1 assembler. And, anyways, it's good to have the `tlutils` package handy. But Mark and many users think that this functionality is good and it took not much time to add this option.]

- **-F** – Force the Unicode encoding: any type of MS encoding specified in the font is ignored and the font is treated like it has Unicode encoding. **WARNING:** this option is intended for buggy fonts which actually are in Unicode but are marked as something else. The effect on the other fonts is unpredictable.

- **-G suboptions** – File generation options. The suboptions may be lowercase or uppercase, the lowercase ones disable the generation of particular files, the corresponding uppercase suboptions enable the generation of the same kind of files. If the result of `ttf2pt1` is requested to be printed on the standard output, the last enabling suboption of **-G** determines which file will be written to the standard output and the rest of files will be discarded. For example, **-G A** will request the AFM file. The suboptions to disable/enable the generation of the files are:

f/F – The font file. Depending on the other options this file will have one of the suffixes `.t1a`, `.pfa` or `.pfb`. If the conversion result is requested on the standard output (‘-’ is used as the output file name) then the font file will also be written there by default, if not overwritten by another suboption of **-G**. **Default: enabled**

a/A – The Adobe font metrics file (`.afm`). **Default: enabled**

e/E – The dvips encoding file (`.enc`). **Default: disabled**

- **-l language[+argument]** – Extract the fonts for the specified language from a multi-language Unicode font. If this option is not used the converter tries to guess the language by the values of the shell variable `LANG`. If it is not able to guess the language by `LANG` it tries all the languages in the order they are listed.

After the plus sign an optional argument for the language extractor may be specified. The format of the argument is absolutely up to the particular language converter. The primary purpose of the argument is to support selection of planes for the multi-plane Eastern encodings but it can also be used in any other way. The language extractor may decide to add the plane name in some form to the name of the resulting font. None of the currently supported languages make any use of the argument yet.

As of now the following languages are supported:

`latin1` – for all the languages using the Latin-1 encoding

`latin2` – for the Central European languages

`latin4` – for the Baltic languages

`latin5` – for the Turkish language

`cyrillic` – for the languages with Cyrillic alphabet

`russian` – historic synonym for `cyrillic`

`bulgarian` – historic synonym for `cyrillic`

`adobestd` – for the AdobeStandard encoding used by TeX

`plane+argument` – to select one plane from a multi-byte encoding

The argument of the “`plane`” language may be in one of three forms:

`plane+pid=<pid>,eid=<eid>`

`plane+pid=<pid>,eid=<eid>,<plane_number>`

`plane+<plane_number>`

`Pid` (TTF platform id) and `eid` (TTF encoding id) select a particular TTF encoding table in the original font. They are specified as decimal numbers. If this particular encoding table is not present in the font file then the conversion fails. The native (“`ttf`”) front-end parser supports only `pid=3` (Windows platform), the FreeType-based (“`ft`”) front-end supports any platform. If `pid/eid` is not specified then the TTF encoding table is determined as usual: Unicode encoding if it’s first or an 8-bit encoding if not (and for an 8-bit encoding the plane number is silently ignored). To prevent the converter from falling back to an 8-bit encoding, specify the Unicode `pid/eid` value explicitly.

`Plane_number` is a hexadecimal (if starts with “`0x`”) or decimal number. It gives the values of upper bytes for which 256 characters will be selected. If not specified, defaults to 0. It is also used as a font name suffix (the leading “`0x`” is not included into the suffix).

NOTE: You may notice that the language names are not uniform: some are the names of particular languages and some are names of encodings. This is because of the different approaches. The original idea was to implement a conversion from Unicode to the appropriate Windows encoding for a

given language. And then use the translation tables to generate the fonts in whatever final encodings are needed. This would allow to pile together the Unicode fonts and the non-Unicode Windows fonts for that language and let the program to sort them out automatically. And then generate fonts in all the possible encodings for that language. An example of this approach is the Russian language support. But if there is no multiplicity of encodings used for some languages and if the non-Unicode fonts are not considered important by the users, another way would be simpler to implement: just provide only one table for extraction of the target encoding from Unicode and don't bother with the translation tables. The latin* "languages" are examples of this approach. If somebody feels that he needs the Type1 fonts both in Latin-* and Windows encodings he or she is absolutely welcome to submit the code to implement it.

WARNING: Some of the glyphs included into the AdobeStandard encoding are not included into the Unicode standard. The most typical examples of such glyphs are ligatures like 'fi', 'fl' etc. Because of this the font designers may place them at various places. The converter tries to do its best, if the glyphs have honest Adobe names and/or are placed at the same codes as in the Microsoft fonts they will be picked up. Otherwise a possible solution is to use the option '**-L**' with an external map.

- **-L file**[+[pid=<pid>,eid=<eid>],[plane]] – Extract the fonts for the specified language from a multi-language font using the map from this file. This is rather like the option '**-I**' but the encoding map is not compiled into the program, it's taken from that file, so it's easy to edit. Examples of such files are provided in maps/adobe-standard-encoding.map, CP1250.map. (**NOTE:** the 'standard encoding' map does not include all the glyphs of the AdobeStandard encoding, it's provided only as an example.) The description of the supported map formats is in the file maps/unicode-sample.map.

Likewise to '**-I**', an argument may be specified after the map file name. But in this case the argument has fixed meaning: it selects the original TTF encoding table (the syntax is the same as in '**-I plane**') and/or a plane of the map file. The plane name also gets added after dash to the font name. The plane is a concept used in the Eastern fonts with big number of glyphs: one TTF font gets divided into multiple Type1 fonts, each containing one plane of up to 256 glyphs. But with a little creativity this concept may be used for other purposes of combining multiple translation maps into one file. To extract multiple planes from a TTF font ttf2pt1 must be run multiple times, each time with a different plane name specified.

The default original TTF encoding table used for the option '**-L**' is Unicode. The map files may include directives to specify different original TTF encodings. However if the pid/eid pair is specified with it overrides any original encoding specified in the map file.

- **-m type=value** – Set maximal or minimal limits of resources. These limits control the the font generation by limiting the resources that the font is permitted to require from the PostScript interpreter. The currently supported types of limits are:

h – the maximal hint stack depth for the substituted hints. The default value is 128, according to the limitation in X11. This seems to be the lowest (and thus the safest) widespread value. To display the hint stack depth required by each glyph in a .t1a file use the script scripts/cntstems.pl.

- **-O suboptions** – Outline processing options. The suboptions may be lowercase or uppercase, the lowercase ones disable the features, the corresponding uppercase suboptions enable the same features. The suboptions to disable/enable features are:

b/B – Guessing of the ForceBold parameter. This parameter helps the Type1 engine to rasterize the bold fonts properly at small sizes. But the algorithm used to guess the proper value of this flag makes that guess based solely on the font name. In rare cases that may cause errors, in these cases you may want to disable this guessing. **Default: enabled**

h/H – Autogeneration of hints. The really complex outlines may confuse the algorithm, so theoretically it may be useful sometimes to disable them. Although up to now it seems that even bad hints are better than no hints at all. **Default: enabled**

u/U – Hint substitution. Hint substitution is a technique permitting generation of more detailed hints for the rasterizer. It allows to use different sets of hints for different parts of a glyph and change these sets as necessary during rasterization (that's why "substituted"). So it should improve the quality of the fonts rendered at small sizes. But there are two catches: First, the X11 library has rather low limit for the font size. More detailed hints increase the file size and thus increase the chance of

hitting this limit (that does not mean that you shall hit it but you may if your fonts are particularly big). This is especially probable for Unicode fonts converted with option ‘-a’, so you may want to use ‘-a’ together with ‘-Ou’. See `app/X11/README` for the description of a patch to X11 which fixes this problem. Second, some rasterizers (again, X11 is the typical example) have a limitation for total number of hints used when drawing a glyph (also known as the hint stack depth). If that stack overflows the glyph is ignored. Starting from version 3.22 `ttf2pt1` uses algorithms to minimizing this depth, with the trade-off of slightly bigger font files. The glyphs which still exceed the limit set by option ‘-mh’ have all the substituted hints removed and only base hints left. The algorithms seem to have been refined far enough to make the fonts with substituted hints look better than the fonts without them or at least the same. Still if the original fonts are not well-designed the detailed hinting may emphasize the defects of the design, such as non-even thickness of lines. So provided that you are not afraid of the X11 bug the best idea would be to generate a font with this feature and without it, then compare the results using the program `other/cmpf` (see the description in `other/README`) and decide which one looks better. **Default: enabled**

o/O – Space optimization of the outlines’ code. This kind of optimization never hurts, and the only reason to disable this feature is for comparison of the generated fonts with the fonts generated by the previous versions of converter. Well, it *_almost_* never hurts. As it turned out there exist some brain-damaged printers which don’t understand it. Actually this feature does not change the outlines at all. The Type 1 font manual provides a set of redundant operators that make font description shorter, such as ‘10 hlineto’ instead of ‘0 10 rlineto’ to describe a horizontal line. This feature enables use of these operators. **Default: enabled**

s/S – Smoothing of outlines. If the font is broken in some way (even the ones that are not easily noticeable), such smoothing may break it further. So disabling this feature is the first thing to be tried if some font looks odd. But with smoothing off the hint generation algorithms may not work properly too. **Default: enabled**

t/T – Auto-scaling to the 1000x1000 Type1 standard matrix. The TTF fonts are described in terms of an arbitrary matrix up to 4000x4000. The converted fonts must be scaled to conform to the Type1 standard. But the scaling introduces additional rounding errors, so it may be curious sometimes to look at the font in its original scale. **Default: enabled**

v/V – Do vectorization on the bitmap fonts. Functionally “vectorization” is the same thing as “auto-tracing”, a different word is used purely to differentiate it from the Autotrace library. It tries to produce nice smooth outlines from bitmaps. This feature is still a work in progress though the results are already mostly decent. **Default: disabled**

w/W – Glyphs’ width corection. This option is designed to be used on broken fonts which specify too narrow widths for the letters. You can tell that a font can benefit from this option if you see that the characters are smashed together without any whitespace between them. This option causes the converter to set the character widths to the actual width of this character plus the width of a typical vertical stem. But on the other hand the well-designed fonts may have characters that look better if their widths are set slightly narrower. Such well-designed fonts will benefit from disabling this feature. You may want to convert a font with and without this feature, compare the results and select the better one. This feature may be used only on proportional fonts, it has no effect on the fixed-width fonts. **Default: disabled**

z/Z – Use the Autotrace library on the bitmap fonts. The results are horrible and **the use of this option is not recommended**. This option is present for experimental purposes. It may change or be removed in the future. The working tracing can be achieved with option **-OV**. **Default: disabled**

- **-p parser_name** – Use the specified front-end parser to read the font file. If this option is not used, `ttf2pt1` selects the parser automatically based on the suffix of the font file name, it uses the first parser in its list that supports this font type. Now two parsers are supported:

`ttf` – built-in parser for the ttf files (suffix `.ttf`)

`bdf` – built-in parser for the BDF files (suffix `.bdf`)

`ft` – parser based on the FreeType-2 library (suffixes `.ttf`, `.otf`, `.pfa`, `.pfb`)

The parser `ft` is **NOT** linked in by default. See `Makefile` for instructions how to enable it. We do no support this parser on Windows: probably it will work but nobody tried and nobody knows how to

build it.

The conversion of the bitmap fonts (such as BDF) is simplistic yet, producing jagged outlines. When converting such fonts, it might be a good idea to turn off the hint substitution (using option **-Ou**) because the hints produced will be huge but not adding much to the quality of the fonts.

- **-u number** – Mark the font with this value as its UniqueID. The UniqueID is used by the printers with the hard disks to cache the rasterized characters and thus significantly speed-up the printing. Some of those printers just can't store the fonts without UniqueID on their disk. The problem is that the ID is supposed to be unique, as its name says. And there is no easy way to create a guaranteed unique ID. Adobe specifies the range 4000000-4999999 for private IDs but still it's difficult to guarantee the uniqueness within it. So if you don't really need the UniqueID don't use it, it's optional. Luckily there are a few millions of possible IDs, so the chances of collision are rather low. If instead of the number a special value 'A' is given then the converter generates the value of UniqueID automatically, as a hash of the font name. (**NOTE:** in the version 3.22 the algorithm for autogeneration of UniqueID was changed to fit the values into the Adobe-specified range. This means that if UniqueIDs were used then the printer's cache may need to be flushed before replacing the fonts converted by an old version with fonts converted by a newer version). A simple way to find if any of the fonts in a given directory have duplicated UniqueIDs is to use the command:

```
cat *.pfa | grep UniqueID | sort | uniq -c | grep -v ' 1 '
```

Or if you use `scripts/convert` it will do that for you automatically plus it will also give the exact list of files with duplicate UIDs.

- **-v size** – Re-scale the font to get the size of a typical uppercase letter somewhere around the specified size. Actually, it re-scales the whole font to get the size of one language-dependent letter to be at least of the specified size. Now this letter is "A" in all the supported languages. The size is specified in the points of the Type 1 coordinate grids, the maximal value is 1000. This is an experimental option and should be used with caution. It tries to increase the visible font size for a given point size and thus make the font more readable. But if overused it may cause the fonts to look out of scale. As of now the interesting values of size for this option seem to be located mostly between 600 and 850. This re-scaling may be quite useful but needs more experience to understand the balance of its effects.
- **-W level** – Select the verbosity level of the warnings. Currently the levels from 0 to 4 are supported. Level 0 means no warnings at all, level 4 means all the possible warnings. The default level is 3. Other levels may be added in the future, so using the level number 99 is recommended to get all the possible warnings. Going below level 2 is not generally recommended because you may miss valuable information about the problems with the fonts being converted.
- **Obsolete option: -A** – Print the font metrics (.afm file) instead of the font on STDOUT. Use **-GA** instead.
- **Very obsolete option:**

The algorithm that implemented the forced fixed width had major flaws, so it was disabled. The code is still in the program and some day it will be refined and returned back. Meanwhile the option name **-f** was reused for another option. The old version was:

-f – Don't try to force the fixed width of font. Normally the converter considers the fonts in which the glyph width deviates by not more than 5% as buggy fixed width fonts and forces them to have really fixed width. If this is undesirable, it can be disabled by this option.

The `.pfa` font format supposes that the description of the characters is binary encoded and encrypted. This converter does not encode or encrypt the data by default, you have to specify the option **-e** or use the `tlasm` program to assemble (that means, encode and encrypt) the font program. The `tlasm` program that is included with the converter is actually a part of the `tlutils` package, rather old version of which may be obtained from

<http://ttf2pt1.sourceforge.net/tlutils.tar.gz>

Note that `tlasm` from the old version of that package won't work properly with the files generated by `ttf2pt1` version 3.20 and later. Please use `tlasm` packaged with `ttf2pt1` or from the new version `tlutils` instead. For a newer version of `tlutils` please look at

<http://www.lcdf.org/~eddi2two/type/>

EXAMPLES

So, the following command lines:

```
ttf2pt1 -e ttffont.ttf t1font
```

```
ttf2pt1 ttffont.ttf - | t1asm >t1font.pfa
```

represent two ways to get a working font. The benefit of the second form is that other filters may be applied to the font between the converter and assembler.

FILES

- `c:/progra~1/Ttf2Pt1/libexec/ttf2pt1/t1asm`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/*`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/scripts/*`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/other/*`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/README`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/FONTS`

SEE ALSO

- the *ttf2pt1_convert(1)* manpage
- the *ttf2pt1_x2gs(1)* manpage
- the *t1asm(1)* manpage
- `ttf2pt1-announce@lists.sourceforge.net`

The mailing list with announcements about ttf2pt1. It is a moderated mailing with extremely low traffic. Everyone is encouraged to subscribe to keep in touch with the current status of project. To subscribe use the Web interface at <http://lists.sourceforge.net/mailman/listinfo/ttf2pt1-announce>. If you have only e-mail access to the Net then send a subscribe request to the development mailing list `ttf2pt1-devel@lists.sourceforge.net` and somebody will help you with subscription.

- `ttf2pt1-devel@lists.sourceforge.net`
`ttf2pt1-users@lists.sourceforge.net`

The ttf2pt1 mailing lists for development and users issues. They have not that much traffic either. To subscribe use the Web interface at <http://lists.sourceforge.net/mailman/listinfo/ttf2pt1-devel> and <http://lists.sourceforge.net/mailman/listinfo/ttf2pt1-users>. If you have only e-mail access to the Net then send a subscribe request to the development mailing list `ttf2pt1-devel@lists.sourceforge.net` and somebody will help you with subscription.

- <http://ttf2pt1.sourceforge.net>

The main page of the project.

<http://www.netspace.net.au/~mheath/ttf2pt1/>

The old main page of the project.

BUGS

It seems that many Eastern fonts use features of the TTF format that are not supported by the ttf2pt1's built-in front-end parser. Because of this for now we recommend using the FreeType-based parser (option **-p ft**) with the "plane" language.

Troubleshooting and bug reports

Have problems with conversion of some font ? The converter dumps core ? Or your printer refuses to understand the converted fonts ? Or some characters are missing ? Or some characters look strange ?

Send the bug reports to the ttf2pt1 development mailing list at `ttf2pt1-devel@lists.sourceforge.net`.

Try to collect more information about the problem and include it into the bug report. (Of course, even better if you would provide a ready fix, but just a detailed bug report is also good). Provide detailed information about your problem, this will speed up the response greatly. Don't just write "this font looks strange after conversion" but describe what's exactly wrong with it: for example, what characters look wrong and what exactly is wrong about their look. Providing a link to the original font file would

be also a good idea. Try to do a little troublehooting and report its result. This not only would help with the fix but may also give you a temporary work-around for the bug.

First, enable full warnings with option ‘**-W99**’, save them to a file and read carefully. Sometimes the problem is with a not implemented feature which is reported in the warnings. Still, reporting about such problems may be a good idea: some features were missed to cut corners, in hope that no real font is using them. So a report about a font using such a feature may motivate someone to implement it. Of course, you may be the most motivated person: after all, you are the one wishing to convert that font. ;-) Seriously, the philosophy “scrath your own itch” seems to be the strongest moving force behind the Open Source software.

The next step is playing with the options. This serves a dual purpose: on one hand, it helps to localize the bug, on the other hand you may be able to get a working version of the font for the meantime while the bug is being fixed. The typical options to try out are: first ‘**-Ou**’, if it does not help then ‘**-Os**’, then ‘**-Oh**’, then ‘**-Oo**’. They are described in a bit more detail above. Try them one by one and in combinations. See if with them the resulting fonts look better.

On some fonts ttf2pt1 just crashes. Commonly that happens because the font being converted is highly defective (although sometimes the bug is in ttf2pt1 itself). In any case it should not crash, so the reports about such cases will help to handle these defects properly in future.

We try to respond to the bug reports in a timely fashion but alas, this may not always be possible, especially if the problem is complex. This is a volunteer project and its resources are limited. Because of this we would appreciate bug reports as detailed as possible, and we would appreciate the ready fixes and contributions even more.

HISTORY

Based on ttf2pfa by Andrew Weeks, and help from Frank Siegert.

Modification by Mark Heath.

Further modification by Sergey Babkin.

The Type1 assembler by I. Lee Hetherington with modifications by Kai-Uwe Herbing.

NAME

ttf2pt1_convert – convenience font conversion script

SYNOPSIS

ttf2pt1_convert [**config-file**]

DESCRIPTION

‘Convert’ is the master conversion script provided with ttf2pt1. When installed into a public directory it’s named **‘ttf2pt1_convert’** to avoid name collisions with the other programs.

If the configuration file is not specified as an argument then the file `‘convert.cfg’` in the current directory is used. This file contains a set of configuration variables. The distribution contains a sample file `‘convert.cfg.sample’`. Please copy it to `‘convert.cfg’`, look inside it and change the configuration variables. The more stable configuration variables, such as the path names of the scripts and encoding files are located in `‘convert’` itself, they are automatically updated when installing **ttf2pt1**.

Put all the TTF fonts you want to convert into some directory (this may be just the directory that already contains all the Windows fonts on a mounted FAT filesystem). If you have fonts in different source encoding then put the fonts in each of the encodings into a separate directory. Up to 10 source directories are supported. If you (in a rather unlikely case) have more source directories then you can make two separate runs of the converter, converting up to 10 directories at a time.

The variables in the configuration file are:

- **SRCDIRS** – the list of directories (with absolute paths) with TTF fonts. Each line contains at least 3 fields: the name of the directory, the language of the fonts in it (if you have fonts for different languages you have to put them into the separate directories) and the encoding of the fonts. Again, if you have some of the TTF typefaces in one encoding, and some in another (say, CP-1251 and KOI-8), you have to put them into the separate source directories. Some lines may contain 4 fields. Then the fourth field is the name of the external map to convert the Unicode fonts into the desirable encoding. This map is used instead of the built-in map for the specified language.
- ***8*** An interesting thing is that some languages have more than one widely used character encodings. For example, the widely used encodings for Russian are IBM CP-866 (MS-DOS and Unix), KOI-8 (Unix and VAX, also the standard Internet encoding), IBM CP-1251 (MS Windows). That’s why I have provided the means to generate the converted fonts in more than one encoding. See the file `encodings/README` for details about the encoding tables. Actually, if you plan to use these fonts with Netscape Navigator better use the aliases `cp-866` instead of `ibm-866` and `windows-1251` instead of `ibm-1251` because that’s what Netscape wants.
- **DSTDIR** – directory for the resulting Type1 fonts. Be careful! This directory gets completely wiped out before conversion, so don’t use any already existing directory for this purpose.
- **DSTENC{language}** – the list of encodings in which the destination fonts will be generated for each language. Each font of that language will be generated in each of the specified encodings. If you don’t want any translation, just specify both **SRCENC** and **DSTENC** as `iso8859-1` (or if you want any other encoding specified in the `fonts.dir`, copy the description of 8859-1 with new name and use this new name for **SRCENC** and **DSTENC**).
- **FOUNDRY** – the foundry name to be used in the `fonts.dir` file. I have set it to `‘fromttf’` to avoid name conflicts with any existing font for sure. But this foundry name is not registered in X11 standards and if you want to get the full standard compliance or have a font server that enforces such a compliance, use `‘misc’`.

The next few parameters control the general behavior of the converter. They default values are set to something reasonable.

- **CORRECTWIDTH** – if the value is set to **YES** then use the converter option **-w**, otherwise don’t use it. See the description of this option in the `README` file.
- **REMOVET1A** – if the value is set to **YES** then after conversion remove the un-encoded `.t1a` font files and the intermediate `.xpfa` font metric files.

- **INSTALLFONTMAP** – a Ghostscript parameter, if the value is set to **YES** then install the entries for the new fonts right into the main Fontmap file. Otherwise just leave the file `Fontmap.ttf` in the Ghostscript configuration directory.
- **HINTSUBST** – if the value is set to **YES** use the option **-H**, otherwise don't use it. This option enables the hint substitution technique. If you have not installed the X11 patch described above, use this option with great caution. See further description of this option in the README file.
- **ENFORCEISO** – if the value is set to **YES** then disguise the resulting fonts as the fonts in ISO Latin1 encoding. Historically this was necessary due to the way the installer scripts created the X11 font configuration files. It is not necessary any more for this purpose. But if you plan to use these fonts with some other application that expects ISO Latin1 encoding then better enable this option.
- **ALLGLYPHS** – if the value is set to **YES** then include all the glyphs from the source fonts into the resulting fonts, even if these glyphs are inaccessible. If it's set to **NO** then include only the glyphs which have codes assigned to them. The glyphs without codes can not be used directly. But some clever programs, such as the Type 1 library from XFree86 3.9 and higher can change the encoding on the fly and use another set of glyphs. If you have not installed the X11 patch described above, use this option with great caution. See further description of the option option **-a** in the README file.
- **GENUID** – if the value is set to **YES** then use the option **-uA** of the converter to generate UniqueIDs for the converted fonts. The standard X11 Type 1 library does not use this ID, so it may only be necessary for the other applications. The script is clever enough to generate different UniqueID for the same font converted to multiple encodings. Also after conversion it checks all the fonts generated during the session for duplicated UniqueID and shows those. Still, this does not guarantee that these UniqueIDs won't overlap with some other fonts. The UniqueIDs are generated as hash values from the font names, so it's guaranteed that if the 'convert' script runs multiple times it will generate the same UniqueIDs during each run. See further description of this option in the README file.
- **GENUID** – if the value is set to **YES** then create the `.pfb` files, otherwise the `.pfa` files. The `.pfb` files are more compact but contain binary data, so you may experience some troubles when transferring them through the network.

The following parameters are used to locate the other scripts and configuration files. By default the scripts do a bit of guessing for them: they search in the **ttf2pt1** installation directory if **ttf2pt1** was installed or otherwise suppose that you are running 'convert' with 'scripts' subdirectory being the current directory.

- **ENCDIR** – directory containing the descriptions of encodings
- **MAPDIR** – directory containing the external map files

Besides that a few parameters are built into the 'convert' script itself. You probably won't need to change them:

- **T1ASM, TTF2PT1, TRANS, T1FDIR, FORCEISO** – paths to the other script

Also there are a few parameters controlling the installation of fonts for Ghostscript. Please look at their description in the Ghostscript section of documentation or in the **ttf2pt1_x2gs(1)** manual page before running 'convert'. If these parameters are set, 'convert' will call the 'x2gs' script automatically to install the newly converted fonts in Ghostscript.

After creating the configuration file run the 'convert' script. Look at the result and the log file in **DSTDIR**.

Add the directory with newly converted fonts to the configuration of X server or font server. For most of the systems this step is very straightforward. For HP-UX it's rather tricky and poorly documented, so the file `FONTS.hpux` gives a short description.

If you don't have the privileges of the root user, you still can configure your private font server. Just use some non-standard port number (see `FONTS.hpux` for an example, except that you won't need all the HP-related stuff on any other system).

FILES

- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/scripts/convert.cfg.sample`

- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/scripts/*`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/README`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/FONTS`
- `c:/progra~1/Ttf2Pt1/share/ttf2pt1/*`
- `c:/progra~1/Ttf2Pt1/bin/ttf2pt1`

SEE ALSO

- the *ttf2pt1(1)* manpage
- the *ttf2pt1_x2gs(1)* manpage
- the *t1asm(1)* manpage

BUGS

Known problems

- One catch is that the X11 Type 1 font library has a rather low limit on the font size. Because of this the fonts with more complicated outlines and the enabled hint substitution may not fit into this limit. The same applies to the fonts with very complicated outlines or with very many glyphs (especially the fonts with over 256 glyphs). So you will need to exercise caution with these options if you plan using these fonts with X11. Some vendors such as HP provide the Type 1 implementation licensed from Adobe which should have no such problem.

But there is a solution even for the generic X11. A patch located in the subdirectory ‘`app/X11`’ fixes this problem as well as some other minor problems. Its description is provided in `app/X11/README`.

To fix the X11 font library, you have to get the X11 sources. I can recommend the ftp sites of the XFree86 project `ftp://ftp.xfree86.org` or of the Open Group `ftp://ftp.x.org`. This patch was made on the sources of XFree86 so you may have better success with applying it to the XFree86 distribution. After you have got the sources, make sure that you can compile them. Then apply the patch as described. Make sure that it was applied properly. Compile the sources again (actually, you need only the fonts library, the fonts server, and possibly the X server). It would be prudent now to save your old font library, font server and, possibly, X server. Then install the new recently compiled versions of these files. Of course, if you know someone who already has compiled these files for the same OS as yours, you can just copy the binary files from him.

Alas, building the X11 system from the source code is not the easiest thing in the world and if you have no experience it can be quite difficult. In this case just avoid the aforementioned features or check each converted font to make sure that it works properly.

- The Type1 font library from the standard X11 distribution does not work on HP-UX (at least, up to 10.01). The font server supplied with HP-UX up to 10.01 is also broken. Starting from HP-UX 10.20 (I don't know about 10.10) they supply a proprietary font library and the converted fonts work fine with it, provided that they are configured properly (see the file `FONTS.hpux`).
- The `fonts.scale` files created by the older versions of the `ttf2pt1` installation program (up to release 3.1) have conflicted with the language definitions of the `xfst` font server and parts of it included into XFree86. To overcome this incompatibility the newer versions create the `fonts.scale` file describing all the fonts as belonging to the `adobe-fontspecific` encoding and the `fonts.alias` file with the proper names. The drawback of this solution is that `xlsfonts` gives the list of twice more fonts. But as a side effect the option **ENFORCEISO** in ‘`convert.cfg`’ is not required for X11 any more.
- The conversion script has no support for Eastern multi-plane fonts. Contribution of such a support would be welcome.

NAME

ttf2pt1_x2gs – font installer for Ghostscript

SYNOPSIS

ttf2pt1_x2gs [**config-file**]

DESCRIPTION

The fonts generated with **ttf2pt1** work fine with Ghostscript by themselves. The script '**x2gs**' (or '**ttf2pt1_x2gs**' when installed into a public directory, to avoid name conflicts with other programs) links the font files from the X11 directory into the Ghostscript directory and automatically creates the description file (**Fontmap**) in Ghostscript format.

If the configuration file is not specified as an argument then the file '**convert.cfg**' in the current directory is used, just like the '**convert**' script does. Indeed, this configuration file is used for both scripts.

The Ghostscript-related parameters in the configuration file are:

DSTDIR – the X11 font directory used by '**x2gs**' as the source of the fonts. This parameter is common with the X11 configuration.

GSDIR – the base directory of Ghostscript. If this parameter is set to an empty string then '**convert**' won't call '**x2gs**'. So if you want to get only the X11 fonts installed then set this parameter to an empty string. This directory may vary on various system, so please check your system and set this value accordingly before running the script.

GSFONTDIR – the font directory of Ghostscript. In the standard Ghostscript installation it's a subdirectory of **GSDIR** but some systems may use completely different directories.

GSCONFDIR – the configuration subdirectory of Ghostscript that contains the **Fontmap** file.

INSTALLFONTMAP – if the value is set to **YES** then install the entries for the new fonts right into the main **Fontmap** file. Otherwise just leave the file **Fontmap.ttf** in the Ghostscript configuration directory.

After preparing the configuration file run the script. It symbolically links all the font files and creates the description file **Fontmap.ttf** in **GSCONFDIR**. After that there are two choices.

If the option **INSTALLFONTMAP** was set to **YES** then the font descriptions are also automatically installed into the master **Fontmap** file. The script is clever enough to detect if it was run multiple times with the same directories and if so it replaces the old **Fontmap** entries with the new ones instead of just accumulating all of them. You may also run it multiple times for multiple X11 directories and all the results will be properly collected in the **Fontmap**. But it's your responsibility to watch that the names of the font files don't overlap. If the X11 font directory gets renamed then you have to remove its font entries from the **Fontmap** and only after that re-run '**x2gs**' for the new directory.

On the other hand if the option **INSTALLFONTMAP** was set to **NO** then go to the **GSCONFDIR** directory and insert the contents of **Fontmap.ttf** into the **Fontmap** file manually. This step may be left manual to make the installation a little bit more safe.

After that you may also want to redefine some of the aliases in **Fontmap** to refer to the newly installed fonts. But the redefinition of the aliases may be dangerous if the width of characters in the new font will be different from the old font. Alas, there is no visible solution of this problem yet.

FILES

- c:/progra~1/Ttf2Pt1/share/ttf2pt1/scripts/convert.cfg.sample
- c:/progra~1/Ttf2Pt1/share/ttf2pt1/scripts/*
- c:/progra~1/Ttf2Pt1/share/ttf2pt1/README
- c:/progra~1/Ttf2Pt1/share/ttf2pt1/FONTS
- c:/progra~1/Ttf2Pt1/share/ttf2pt1/*
- c:/progra~1/Ttf2Pt1/bin/ttf2pt1

SEE ALSO

- the *ttf2pt1(1)* manpage
- the *ttf2pt1_convert(1)* manpage
- the *tlasm(1)* manpage