Copyright (C) 1990, 1991 Aladdin Enterprises. All rights reserved.

This file is part of Ghostscript.

Ghostscript is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY. No author or distributor accepts responsibility
to anyone for the consequences of using it or for whether it serves any
particular purpose or works at all, unless he says so in writing. Refer
to the Ghostscript General Public License for full details.

Everyone is granted permission to copy, modify and redistribute
Ghostscript, but only under the conditions described in the Ghostscript
General Public License. A copy of this license is supposed to have been
given to you along with Ghostscript so you can know your rights and
responsibilities. It should be in a file named COPYING. Among other
things, the copyright notice and this notice must be preserved on all
copies.

- -

This file, fonts.doc, describes the fonts and font facilities supplied
with Ghostscript.

For an overview of Ghostscript and a list of the documentation files, see
README.

About Ghostscript fonts

The fonts included with Ghostscript come in several parts:

- Font data in files *.gsf: each file defines one
  (transformable) font specified in outline form.

- BuildChar procedures in gs_fonts.ps: these provide the
  algorithms for interpreting the data in the .gsf files.

- The Fontmap file: this relates Ghostscript font names to .gsf
  file names.

Currently, most of the fonts supplied with Ghostscript are based on
various public domain bitmap fonts, primarily the ones supplied with the
X11 distribution from MIT, and on the public domain Hershey fonts. The
fonts are distributed in the file `ghostscript-N.NNfonts.tar.Z'. The
bitmap-derived fonts include the usual Helvetica, Times-Roman, and so on;
see the file `Fontmap' for the complete list, in the usual roman, italic,
bold, and bold italic styles (for the most part). The Hershey fonts, on
the other hand, are quite different from traditional ones; the file
`hershey.doc' describes them in more detail.

There is also a single rather heavy home-grown font called Ugly. This
font is the file `uglyr.gsf' in the Ghostscript source distribution.

The file gs_fonts.ps, which is loaded as part of Ghostscript
initialization, arranges to load fonts on demand using the information
from Fontmap. If you want to preload all of the known fonts, invoke the
procedure

```
loadallfonts
```

This is not done by default, since the fonts occupy about 50K each and there

are a lot of them.

Ghostscript fonts are actually ordinary Ghostscript programs: they use the extension .gsf instead of .ps simply to be informative. This convention is only embodied in the Fontmap file: there is no code that knows about it.

If you want to try out the fonts, prfont.ps contains code for printing a sampler. Load this program, by including it in the gs command line or by invoking

```
(prfont.ps) run
```

and then to produce a sampler of a particular font, invoke

```
/fontName DoFont
```

e.g.

```
/Times-Roman DoFont
```

Contents of fonts

A Ghostscript font is a dictionary with a standard set of keys as follows. The keys marked with a * have the same meanings as in P*stScr*pt fonts; those marked with # have the same meanings as in Adobe Type 1 fonts. Note that FontName is required; StrokeWidth is required for all stroked or outlined fonts; and Metrics is not currently supported.

* - FontMatrix <array>: the transformation from character coordinates to user coordinates.

* - FontType <integer>: the type of the font, either 1 or 3.

* - FontBBox <array>: the bounding box of the font.

* - Encoding <array>: the map from character codes to character names.

* - FontName <name>: the name of the font.

* - PaintType <integer>: an indication of how to interpret the character description from CharInfo.

* - StrokeWidth <number>: the stroke width for outline fonts.

* - FontInfo <dictionary>: additional information about the font (optional, not used by the standard Ghostscript software).

* - UniqueID <integer>: a unique number identifying the font.

* - BuildChar <procedure>: the procedure for showing a character (not required in type 1 fonts).

\# - CharStrings <dictionary>: the map from character names to character descriptions (relevant only in type 1 fonts).

\# - Private <dictionary>: additional information used by the algorithms for rendering outlines fonts (relevant only in type 1 fonts).

The format of values in the CharStrings and Private dictionaries are described in the Adobe Type 1 Font Format book.

Adding your own fonts

Ghostscript can use any Type 1 or Type 3 font that is acceptable to other PostScript language interpreters. Ghostscript also provides a way to construct a Type 1 font from a bitmap font in BDF format, which is a popular format in the Unix world.

If you want to add fonts of your own, you must edit Fontmap to include an entry for your new font at the end. The format for entries is documented in the Fontmap file. Since later entries in Fontmap override earlier entries, any fonts you add will supersede the corresponding fonts supplied with Ghostscript. Note that the font name in the Fontmap entry must be the same as the FontName in the font itself.

In the PC world, Type 1 fonts are customarily given names ending in .PFA or .PFB. Ghostscript can use these directly; you just need to make the entry in Fontmap. If you are going to use a commercial Type 1 font (such as fonts obtained in conjunction with Adobe Type Manager) with Ghostscript, please read carefully the license that accompanies the font; Aladdin Enterprises and the Free Software Foundation take no responsibility for any possible violations of such licenses.

If you want to convert a BDF file to a scalable outline, use the program bdftops.ps (and invoking shell script bdftops.bat or bdftops). Run the shell command

```
bdftops <BDF_file_name> [<AFM_file1_name> ...] <your_gsf_file_name>
  <font_name> <uniqueID> [<encoding_name>]
```

e.g.,

```
bdftops pzdr.bdf ZapfDingbats.afm pzdr.gsf ZapfDingbats 4100000
```

Then make an entry for the .gsf file in Fontmap as described above. You may find it helpful to read, and to add an entry to, the fonts.mak file, which is a makefile for converting the standard Ghostscript fonts.

Precompiling fonts

You can compile any Type 1 font into C and link it into the Ghostscript executable. (Type 1 fonts include any font whose name ends with .pfa or .pfb, and it also includes all the Ghostscript .gsf fonts except for the Hershey fonts.) This doesn't have any effect on rendering speed, but it eliminates the time for loading the font dynamically, which may make a big difference in total rendering time, especially for multi-page documents. (Because of RAM and compiler limitations, you should only use compiled fonts on MS-DOS systems if you are using a 32-bit compiler such as Watcom C/386 or djgpp; you will run out of memory if you use compiled fonts with the Borland compiler.) Fonts that have been precompiled and linked in this way do not need to appear in the Fontmap, although if they do appear there, no harm is done.

The utility for precompiling fonts is called font2c. Note that font2c is a PostScript language program, so you must have Ghostscript already running to be able to run font2c; you must also have entries in the Fontmap for the fonts you want to compile.) For example, to precompile the Times-Italic font,

```
font2c Times-Italic ptmri.c
```

where the first argument is the font name and the second is the name of the .c file. You can use any file name you want, as long as it ends in

.c. It doesn't have to be limited to 8 characters, unless your operating system requires this. We suggest that you use names xxxx.c, where xxxx.gsf or xxxx.pfa is the name of the font file in the Fontmap file, just so you don't have to keep track of another set of names. (If you are running on a VMS platform, or another platform where the C compiler has a limit on the length of identifiers, you must do something slightly more complicated; see the section 'Platforms with identifier length limits' below. Also, on VMS, you must put quotes "" around the font name so that the VMS command processor doesn't convert the name to lower case.)

Besides running font2c, you must arrange things so that the file will be compiled from C to machine code, and linked into the executable. All environments except VMS use the same procedure for this, which we will now describe. For VMS environments, the necessary information is contained in comments in the command files (vms-cc.mak and vms-gcc.mak); if you are using Ghostscript on VMS, ignore the rest of this subsection.

First, you must add the compiled fonts "feature" to your platform-specific makefile. On MS-DOS systems, you edit tc.mak, bc.mak, bcwin.mak, msc.mak, or watc.mak; on Unix systems, you edit makefile. Find the definition of FEATURE_DEVS in the makefile, e.g.,

```
      FEATURE_DEVS=filter.dev dps.dev
```

Add ccfonts.dev on the end, e.g.,

```
      FEATURE_DEVS=filter.dev dps.dev ccfonts.dev
```

Next, you must add the specific fonts to the generic makefile. On MS-DOS systems, you edit gs.mak; on Unix systems, you edit makefile. Find the line in the relevant makefile that says

```
      ccfonts1_=ugly.$(OBJ)
```

Edit this to add your compiled font file names, e.g.,

```
      ccfonts1_=ugly.$(OBJ) ptmri.$(OBJ)
```

If the line gets too long, add another line of the same form, e.g.,

```
      ccfonts2_=ptmb.$(OBJ)
```

Just below this, you will find a line that says

```
      ccfonts1=Ugly
```

Add your own fonts to the end of this line, e.g.,

```
      ccfonts1=Ugly Times_Italic
```

Notice that you must replace `-' by `_' in the font name. Again, if the line gets too long, add another line of the same form, e.g.,

```
      ccfonts1=Ugly
      ccfonts2=Times_Italic
```

Now find the lines that say

```
      ugly.$(OBJ): ugly.c $(CCFONT)
            $(CCCF) ugly.c
```

Add a similar pair of lines for each font, separating these entries from the existing entries and from each other by a blank line. (The makefile includes lines for a few more fonts than this already.) In our example,

```
      ugly.$(OBJ): ugly.c $(CCFONT)
            $(CCCF) ugly.c

      ptmri.$(OBJ): ptmri.c $(CCFONT)
            $(CCCF) ptmri.c
```

Finally, run `make'. The executable will now include the fonts you added. They will be present in FontDirectory when Ghostscript starts up.

Note that ugly.c, ncrr.c, etc. are not supplied with the Ghostscript fileset, since they are quite large and can easily be recreated using the font2c program as described above.

Platforms with identifier length limits

On some platforms, the C compiler and/or linker have a limit on the number of significant characters in an identifier. On such platforms, you must do a little extra work.

Let N be the maximum number of significant characters in an identifier (typically 31). For each font whose name is longer than N-5 characters, pick an arbitrary identifier that we will call the "short name". This can be any string you want, as long as it contains only letters, digits, and underscores; is no longer than N-5 characters; and is not the same as any other font name or short name. A good choice for this would be to use the name of the C file. (There is no harm in doing this for fonts with names shorter than N-5 characters, it's just not necessary.)

You must do two different things for fonts that require a short name. First, you must supply the short name as a third argument to the font2c program. For example, to compile NewCenturySchlbk-BoldItalic using the short name "pncbi",

```
font2c NewCenturySchlbk-BoldItalic pncbi.c pncbi
```

Then when you add the font to the gsaddmod line in the makefile, use the short name, not the actual font name, e.g.,

```
ccfonts2=pncbi
```

instead of

```
ccfonts2=NewCenturySchlbk_BoldItalic
```

Everything else is as described above.

This procedure doesn't change the name of the font in the Fontmap, or as seen from within Ghostscript; it's just a workaround for a limitation of some older compilers.