- -

This file, make.doc, describes how to install Ghostscript, and how to
build Ghostscript executables from source.

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

****
******** Installing Ghostscript
****

To install the interpreter, you need:
- The interpreter executable:
    - On MS-DOS and VMS systems, gs.exe.
    - On MS-DOS systems, if you are using the Watcom compiler,
      the DOS extender, dos4gw.exe.
    - On Unix systems, gs.
- The interpreter initialization files: gs_*.ps.
- The font map: Fontmap.
- The default font: uglyr.gsf.

See use.doc for a description of the search algorithm used to find these
files.

You do not need any of these files when using the library; however, the
library currently provides no way to install fonts. This is obviously
ridiculous and will be fixed sometime in the future.

****
******** Building Ghostscript from source
****

Ghostscript is generally distributed in the form of a compressed tar file.
When unpacked, this file puts all the Ghostscript files in a directory
called gs. Ghostscript is also available in the form of PC-compatible ZIP
files.

Ghostscript is described by a collection of several makefiles:

gs.mak - a generic makefile used on all platforms (except VMS).

devs.mak - a makefile listing all the device drivers.
*.mak - the makefiles for specific platforms.

You may need to edit the platform-specific makefile if you wish to change any of the following options:

- The default search path(s) for the initialization and font files (macro GS_LIB_DEFAULT);

- The debugging options (macro TDEBUG);

- The set of device drivers to be included (DEVICE_DEVS and DEVICE_DEVS1..9 macros);

- The set of optional features to be included (FEATURE_DEVS macro).

The platform-specific makefile will include comments describing all of these items except the DEVICE_DEVS options. The DEVICE_DEVS options are described in devs.mak, even though the file that must be edited is the platform-specific makefile.

The makefiles distributed with Ghostscript define these options as follows:

- GS_LIB_DEFAULT: on Unix systems, /usr/local/lib/ghostscript and /usr/local/lib/ghostscript/fonts; on MS-DOS systems, C:\GS.

- TDEBUG: no debugging code included in the build.

- DEVICE_DEVS*: platform-specific, see below.

- FEATURE_DEVS: platform-specific.

There are also platform-specific options described below under the individual platforms. See the "Options" section near the beginning of the relevant makefile for more information.

If you are including a dot-matrix printer driver, you may wish to customize the default resolution parameters in devs.mak.

To build the interpreter, you need all the .h and .c files (and .asm files for MS-DOS) included in the distribution, as well as the makefiles.

The command

```
make clean
```

removes all the files created by the build process (relocatables, executables, and miscellaneous scratch files). If you want to save the executable, you should move it to another directory first.

****
******** How to build Ghostscript from source (MS-DOS version) ********
****

To find out what devices the makefiles distributed with Ghostscript select for inclusion in the executable, find the lines in the appropriate makefiles of the form

```
FEATURE_DEVS=<list of features>
```

and

```
DEVICE_DEVS=<list of devices>
```

(similarly DEVICE_DEVS1... up to DEVICE_DEVS9)
The relevant makefiles are:
Turbo C: tc.mak
Turbo C++/Borland C++, MS-DOS: bc.mak
Borland C++, MS Windows: bcwin.mak
Microsoft C/C++ 7.0, MS-DOS: msc.mak
Watcom C/386, MS-DOS: watc.mak
The options were chosen to strike a balance between RAM consumption and likely usefulness. (Turbo C is limited to 640K and does not support code overlaying; Borland C++ is limited to 640K, but supports code overlaying under MS-DOS; Watcom C/386 is not limited to 640K.)

To build Ghostscript, you need MS-DOS version 3.3 or later, and a (Borland) Turbo C/C++, Borland C/C++, Microsoft C/C++ (version 7), or Watcom C/386 development system. Details are given below.

As noted above, the default configuration generates an executable that assumes the directory where 'make' was run should be the final default directory for looking up the Ghostscript initialization and font files.

To build the Ghostscript executable, all you need to do is give the command
make
You must have COMMAND.COM in your path to build Ghostscript.

There is a special 'make' target that simply attempts to compile all the .c files in the current directory. Some of these compilations will fail, but the ones that succeed will go considerably faster, because they don't individually pay the overhead of loading the compiler into memory. So a good strategy for building the executable for the first time, or after a change to a very widely used .h file, is:
make begin
and then
make
to do the compilations that failed the first time.

Note: if you get the Ghostscript sources from a Unix 'tar' file and unpack the file on a MS-DOS machine, the files will all have linefeed instead of carriage return + linefeed as the line terminator, which may make the C compiler unhappy. I don't know the simplest way to fix this: just reading each file into an editor and writing it back out again may be sufficient. You will probably have to do this to the .c, .h, and .bat files.

Borland environment

To compile Ghostscript with the Borland environment, you need either Turbo C (version 2.0 or later) or Turbo C++ or Borland C++ (version 1.0 or later); specifically, the compiler, 'make' utility, and linker. You also need either the Borland assembler (version 1.0 or later) or the Microsoft assembler (version 4.0 or later). Before compiling or linking, you should execute
echo !include "tc.mak" >makefile
(for Turbo C and MS-DOS), or
echo !include "bc.mak" >makefile
(for Turbo C++ or Borland C++ and MS-DOS), or
echo !include "bcwin.mak" >makefile
(for Turbo C++ or Borland C++ and Microsoft Windows), or

Besides the source files and the makefiles, you need:
      turboc.cfg (the flags and switches for Turbo C)
      gs.tr (the linker commands for the interpreter)
      *.bat (a variety of batch files used in the build process)

There are extensive comments in the aforementioned .mak files regarding various configuration parameters. If your configuration is different from the following, you should definitely read those comments and see if you want or need to change any of the parameters:
      - The compiler files are in c:\tc (for Turbo C) or c:\bc (for Turbo C++ or Borland C++) and its subdirectories.
      - You are using the Borland assembler (tasm).
      - You want an executable that will run on any PC-compatible, regardless of processor type (8088, 8086, V20, 80186, 80286, V30, 80386, 80486) and regardless of whether a math coprocessor (80x87) is present.

NOTE: Borland C++ 3.0 has two problems that affect Ghostscript (these problems are fixed in Borland C++ 3.1):

      - The assembler, tasm, often crashes when attempting to assemble gdevegaa.asm. If this happens, try again, or use another assembler (e.g., an older version of tasm) if you have one, or set USE_ASM=0 in the makefile.

      - The math library for Microsoft Windows, mathwl.lib, has a bug that causes floating point numbers to print incorrectly. Contact Borland for a corrected version.

If you are compiling Ghostscript with Turbo C++ 1.0, remove the `.swap' directive from bc.mak, and use the -s switch on the `make' command line. (All later versions of the Borland environment recognize this directive.)

Note that although the Microsoft Windows version of Ghostscript will run under Windows 3.0, it uses routines from the Windows 3.1 SDK, so you need the Windows 3.1 SDK and header files to compile it. In practice, this means that you need Borland C++ 3.1; Borland C++ 3.0 doesn't include the necessary headers.

Microsoft environment

To compile Ghostscript with the Microsoft environment, you need Microsoft C/C++ 7.0 or later with its associated `nmake' utility and linker. Before compiling or linking, you should execute
      echo !include msc.mak >makefile

Besides the source files and the makefiles, you need:
      gs.tr (the linker commands for the interpreter)
      *.bat (a variety of batch files used in the build process)

We have found the Microsoft `nmake' program to be very unreliable; it may crash partway through the build process with a variety of error messages, or no error message.

We have not been able to test the Microsoft compiler in a wide variety of configurations, because it is so much slower than the Borland compiler (about 5 times) and because there is apparently no way to get it to write its error messages to a file. If you

encounter problems with the Microsoft environment, please contact
Aladdin Enterprises.

Watcom environment

To avoid annoying messages from the DOS extender, add the line
```
      set DOS4G=quiet
```
to your autoexec.bat file.

To compile Ghostscript with the Watcom C/386 compiler, you need to create
a makefile by executing

```
      echo !include watc.mak >makefile
```

To build Ghostscript, execute

```
      wmake -u
```

****
******** How to build Ghostscript from source (Unix version) ********
****

The makefile distributed with Ghostscript selects the following devices
for inclusion in the build:
      Display: X Windows driver.
      File output: pbm, pbmraw, pgm, pgmraw, ppm, and ppmraw drivers.

Before compiling or linking, you should execute

```
      ln -s unix-cc.mak makefile
or    ln -s unix-gcc.mak makefile
or    ln -s unix-ansi.mak makefile
```

(if your Unix system doesn't support symbolic links, omit the -s switch)
depending on whether your C compiler is a standard Kernighan & Ritchie C
compiler, gcc being used in ANSI mode, or an ANSI C compiler other than
gcc respectively. (If you want to use gcc in non-ANSI mode, use
unix-cc.mak and define the CC macro to refer to gcc.)

The unix-*.mak files are actually generated mechanically from *head.mak,
*tail.mak, gs.mak, and devs.mak. If for some reason your copy of
Ghostscript doesn't include the unix-*.mak files, invoke the
```
      tar_cat
```
shell script to construct them.

If the X11 client header files are located in some directory which your
compiler does not automatically search, you must change the XINCLUDE macro
the makefile to include a specific -I switch. See the comment preceding
XINCLUDE in the makefile.

The only important customization of the X11 driver is the choice of
whether or not to use a backing pixmap. If you use a backing pixmap,
Ghostscript windows will redisplay properly when they are covered and
exposed, but drawing operations will go slower. This choice is controlled
by a line in the file gdevxini.c that says
```
      private int use_backing = 1;
```
Changing this line to read
```
      private int use_backing = 0;
```

will disable the use of a backing pixmap. However, portions of the Ghostscript window may not be properly redrawn after the window is restored from an icon or exposed after being occluded by another window.

Some versions of the X server do not implement tiling properly. This will show up as broad bands of color where dither patterns should appear. If this happens, in the file gdevx.c, change

```
      private int use_XSetTile = 1;
```

to

```
      private int use_XSetTile = 0;
```

and recompile. The result will run a lot slower, but the output should be correct. If this fixes the problem, report it to whoever made your X server.

Similarly, some versions of the X server do not implement bitmap/pixmap displaying properly. This may show up as white or black rectangles where characters should appear, or characters may appear in "inverse video" (e.g., white on a black rectangle). If this happens, it may help you to change, in the file gdevx.c,

```
      private int use_XPutImage = 1;
```

to

```
      private int use_XPutImage = 0;
```

and recompile. Again, there will be a serious performance penalty; again, if this fixes the problem, notify the supplier of your X server.

Currently Ghostscript is set up to compile and link in a generic Unix environment. Some Unix environments may require changing the LDFLAGS macro in the makefile.

All you need to do to make an executable is invoke the shell command

```
      make
```

Ghostscript uses ANSI syntax for function definitions. Because of this, when compiling with cc, it must preprocess each .c file to convert it to the older syntax defined in Kernighan and Ritchie, which is what most current Unix compilers (other than gcc) support. This step is automatically performed by a utility called ansi2knr, which is included in the Ghostscript distribution. The makefile automatically builds ansi2knr.

The ansi2knr preprocessing step is included in the makefile rule for compiling .c files. ansi2knr creates a file called _temp_.c to hold the converted code. If you want to change this name for some reason, it is defined in unix-cc.mak.

Platform-specific notes

386 Unix:

gcc versions older than 1.38 on Intel 80386 systems do not compile Ghostscript correctly using the -O option. Do not use -O in these environments.

gcc 1.39 under 386BSD has a bug that causes float-to-integer conversions to compile incorrectly. Do not use this version of gcc.

X11R5 may need #include <stddef.h> in x_.h.

Also see below regarding System V platforms.

Apollo:

You must run the compiler in ANSI-compatible mode (i.e., set AK= <null string> in the makefile); otherwise, it gives incorrect error

messages for any function declared as returning a float value.
	The Apollo compiler may not compile Ghostscript correctly. If you get unexpected crashes at run time, use gcc.

Convex:
	Use unix-ansi.mak. Do not invoke optimization (-O1): there are compiler bugs that lead to incorrect code. Set CFLAGS to
	-fn -tm -no c1

DEC (Ultrix):
	Many versions of DEC's X server (DECwindows) have bugs that require setting use_XPutImage or use_XSetTile to 0, as described above.

GNU make (any platform):
	GNU make 3.59 can't handle the final linking step in some cases; use the platform's standard make (e.g., /bin/make) if this happens.

H-P 700 series:
	Use the compiler flags -Aa +O3 (*not* -O).

ISC Unix:
	For ISC Unix with gcc, an appropriate make invocation is:
	make XCFLAGS="-D__SVR3 -posix" LDFLAGS="-shlib -posix" \
	 EXTRALIBS="-linet -lnsl_s"
If this doesn't work for you, try removing the -shlib. ISC Unix may also need one or more of the following in EXTRALIBS: -lpt, -lc_s.
See also under "386 Unix" above.

MIPS:
	There is apparently a bug in the MIPS C compiler which causes gxdither.c to compile incorrectly if optimization is enabled (-O). Until a work-around is found, do not use -O with the MIPS C compiler.

NeXT:
	Use unix-gcc.mak, but change the name of the compiler (CC=) from gcc to cc. Also, include -D_NEXT_SOURCE in CFLAGS, and change the two occurrences of sys/time.h to ansi/time.h. You may also find it useful to add the following line to Fontmap:
	 /Ohlfs /Courier ;

RS/6000:
	Due to compiler bugs, most of the obvious ways to compile Ghostscript on the RS/6000 don't work. You should use the following command line, or a similar one with a different compiler (see below):

	make -f unix-ansi.mak CC=c89 \
	XINCLUDE=-I/usr/misc/X11/include XLIBDIRS=-L/usr/misc/X11/lib

You must also edit the makefile (unix-ansi.mak or unix-cc.mak) to define CFLAGS as -DSYSV -D_POSIX_SOURCE, and to change INSTALL to /usr/ucb/install. (If -DSYSV produces a complaint about the functions index and rindex not being defined, try removing it.)

Even beyond all this, many people have trouble with Ghostscript on the RS/6000. The usual symptom is that it compiles and links OK, but produces garbaged output on the screen. The problem may be related to particular versions of AIX or the X server; we don't have enough information at this time. The following combinations of AIX and X are known to fail:

AIX 3.1.5, MIT X11R4, c89, unix-ansi.mak
AIX 3.2.0 or 3.2.1, c89 or xlc
The following combinations are known to work:
AIX 3.1.5, MIT X11R5, c89, unix-ansi.mak
AIX 3.2, MIT X11R5, xlc 1.2.0.9, unix-ansi.mak
AIX 3.2, c89, unix-ansi.mak -- but you must compile without optimization (no -O)
AIX 3.2.0 or 3.2.1, cc, unix-cc.mak, no -O
It may be that just including -Dconst= in the command line is sufficient to get around the bugs in the AIX 3.1.5 compiler.

Apparently some (but not all) releases of the C library declare the hypot function: if the declaration in math_.h produces an error message, try removing it. Also, the IBM X11R3 server is known to be buggy: use the MIT X server if possible.

SCO Unix/Xenix:

The SCO Unix C compiler apparently can't handle the Pn macros in std.h. If you get strange compilation errors on SCO Unix, see if you can get a compiler fix from SCO. Meanwhile, to use gcc with SCO ODT, see gcc-head.mak for the appropriate switch settings. See also under "386 Unix" above.

gcc 2.3.3 produces code that causes a core dump on machines that don't have hardware floating point, because of a bug in SCO's floating point emulator. Use a different compiler on these machines.

If you aren't using the X11 driver, you need to add -lsocket to the linker command (near the end of the unix-*.mak file) in order to get the date/time functions linked in.

If you want to use direct frame buffer addressing instead of X Windows, include the relevant frame buffer device(s) (ega.dev, vga.dev, etc.) and change gdevevga.c to gdevsco.c as indicated in devs.mak. Note: this does not work with SuperVGA displays, except for 800x600x16 mode.

If your compiler accepts the -Xt and -Xa switches, use -Xt. Even though this causes the compiler to use incorrect rules for computing the result types of << and >>, -Xa enables "optimizations" that produce incorrect code.

For SCO ODT 2.0, in addition to -D__SVR3 and -DSYSV, you need to specify -Dsco, -DUSG, and -DMALLOC_0_RETURNS_NULL. For SCO ODT, you need EXTRALIBS=-lX11 -lsocket -lmalloc, or maybe only -lsocket (depending on the version), and for SCO ODT 2.0, you also need to specify -lc_s. For SCO Xenix, you need EXTRALIBS=-lmalloc.

For all SCO systems, set XINCLUDE= and XLIBDIRS=.

Sun:

The Sun unbundled C compiler (SC1.0) doesn't compile Ghostscript properly if the -fast option is selected: Ghostscript core-dumps in build_gs_font. Use -g, or use gcc.

The Sun version of dbx often gives up with an error message when trying to load Ghostscript. If this happens, use gdb instead. (gdb is more reliable than dbx in other ways as well.)

Solaris 2.2 may require setting EXTRALIBS=-lsocket.

In SunOS 4.1.[23], you may get these undefined symbols when linking:

```
_get_wmShellWidgetClass
_get_applicationShellWidgetClass
```

Compiling "-Bstatic -lXmu -Bdynamic" appears to work. To solve the problem if you are using OpenWindows 3.0 (X11R4-based Xt), please contact your local Sun office and request the following patches:

Patch i.d. Description

100512-02 4.1.x OpenWindows 3.0 libXt Jumbo patch
100573-03 4.1.x OpenWindows 3.0 undefined symbols when using shared libXmu

A source patch for use with the MIT X11R4 libraries was developed by Conrad Kimball (cek@sdc.boeing.com); it retrofits into R4 some fixes made in R5 to get around this problem. The patch is on export in [1/93]
contrib/X11R4_sunos4.1.2_patch_version3.Z

System V Unix platforms:

If you are using a stock System V platform that lacks rename and gettimeofday, change PLATFORM=unix_ in the makefile to PLATFORM=sysv_.

You will probably need to change the definition of INSTALL (near the beginning of the makefile) from install to /usr/ucb/install.

****
******** How to build Ghostscript from source (VAX/VMS version) ********
****

The files VMS-CC.MAK and VMS-GCC.MAK are VMS DCL command files which build Ghostscript from scratch using either the DEC C compiler, CC, or the Free Software Foundation's GNU C compiler, GCC. Accordingly, you must have one of these two compilers installed in order to build Ghostscript. (Other C compilers may work: CC and GCC are the only two compilers tested to date.) These two command files build and store the Ghostscript library in the object library GS.OLB. If you have DECwindows (X11) installed on your system, the executable images GS.EXE, GT.EXE, and XLIB.EXE will also be built.

Some environments use the DWTLIBSHR library for providing the X Windows intrinsics, and some use the XTSHR library. XTSHR is newer, and is part of the DECwindows/Motif product. However, DEC is still distributing versions of VMS with DWTLIBSHR. If your environment uses XTSHR, replace DWTLIBSHR in the list of link libraries with XTSHR.

The only important customization of the X11 driver is the choice of whether or not to use a backing pixmap. If you use a backing pixmap, Ghostscript windows will redisplay properly when they are covered and exposed, but drawing operations will go slower. This choice is controlled by the line in the file gdevx.c that reads

```
private int use_backing = 1;
```

Changing this line to read

```
private int use_backing = 0;
```

will disable the use of a backing pixmap. However, portions of the Ghostscript window may not be properly redrawn after the window is restored from an icon or exposed after being occluded by another window.

Many versions of DEC's X server (DECwindows) have bugs that require setting use_XPutImage or use_XSetTile to 0, as described above. These show up as broad bands of color where dither patterns should appear, or characters displayed white on top of black rectangles or not displayed at all. If this happens, change use_XSetTile or use_XPutImage to 0 as described above. The result will run a lot slower, but the output will be correct. Report the problem to DEC, or whoever supplied your X server.

Ghostscript uses ANSI syntax for function definitions. Thus, when using the DEC C compiler, each .C file is converted to the older C syntax defined in the first edition of Kernighan and Ritchie and stored in a .CC file.

This step is performed by VMS-CC.MAK using the ansi2knr utility included in the Ghostscript distribution. If you are building a debuggable configuration, the .CC files will be left behind by VMS-CC.MAK for use by the VMS Debugger; otherwise, they will be deleted.

If you have DEC's C compiler, issue the DCL command

```
$ @VMS-CC.MAK
```

to build Ghostscript. If you have GNU C, issue the DCL command

```
$ @VMS-GCC.MAK
```

to build Ghostscript.

The option "DEBUG" may be specified with either command file in order to build a debuggable Ghostscript configuration:

```
$ @VMS-CC.MAK DEBUG    -or-   $ @VMS-GCC.MAK DEBUG
```

In order to specify switches and file names when invoking the interpreter, define GS as a foreign command:

```
$ GS == "$disk:[directory]GS.EXE"
```

where "disk" and "directory" specify the disk and directory where Ghostscript is located. For instance,

```
$ GS == "$DUA1:[GHOSTSCRIPT]GS.EXE"
```

To allow the interpreter to be run from any directory, define the logical GS_LIB which points to the Ghostscript directory

```
$ DEFINE GS_LIB disk:[directory]
```

This allows Ghostscript to locate its initialization files stored in the Ghostscript directory -- see use.doc for further details. Finally, to invoke the interpreter, merely type GS. Although DCL normally converts unquoted parameters to upper case, C programs receive their parameters in lower case. That is, the command

```
$ GS -Isys$login:
```

passes the switch "-isys$login" to the interpreter. To preserve the case of switches, enclose them in double quotes; e.g.,

```
$ GS "-Isys$login:"
```

If you add compiled fonts to your system as described in the fonts.doc file, you must C-compile them in an environment that includes some definitions from vms-cc.mak. Find the section of vms-cc.mak with the comment "Give ourself a healthy search list for C include files" and execute the immediately following DEFINE commands before C-compiling the fonts.

****
******** A guide to the files ********
****

General

There are very few machine dependencies in Ghostscript. A few of the .c files are machine-specific. These have names of the form

gp_<platform>.c

specifically

- gp_dosfb.c (all MS-DOS platforms)
- gp_msdos.c (all MS-DOS platforms)
- gp_itbc.c (MS-DOS, Borland compilers)
- gp_iwatc.c (MS-DOS, Watcom or Microsoft compiler)
- gp_unix.c (all Unix)
- gp_sysv.c (System V Unix)
- gp_vms.c (VMS)

There are also some machine-specific conditionals in files with names <something>_.h. If you are going to extend Ghostscript to new machines or operating systems, you should check the *_.h files for ifdef's on things other than DEBUG, and you should probably count on making a new makefile and a new gp_ file.

Library

Files beginning with gs, gx, or gz (both .c and .h), other than gs.c and gsmain.c, are the Ghostscript library. Files beginning with gdev are device drivers or related code, also part of the library. Other files beginning with g are library files that don't fall neatly into either the kernel or the driver category.

Interpreter

gs.c is the main program for the language interpreter.

Files beginning with z are Ghostscript operator files. The names of the files generally follow the section headings of the operator summary in section 6.2 of the PostScript manual.

.c files beginning with i, and .h files not beginning with g, are the rest of the interpreter. See the makefile for a little more information on how the files are divided functionally.

There are a few files that are logically part of the interpreter, but that are potentially useful outside Ghostscript, whose names don't begin with either g, z, or i:

s*.c (a flexible stream package, including the Level 2 PostScript 'filters' supported by Ghostscript);