Web2c

for version 2021
February 2021

Karl Berry
Olaf Weber
`https://tug.org/web2c`

This file documents the installation and use of the programs in Web2c, an implementation of Donald Knuth's TeX system.

Table of Contents

1 Introduction

This manual corresponds to version 2021 of Web2c, released in February 2021.

Web2c is the name of a TeX implementation, originally for Unix, but now also running under various operating systems. By *TeX implementation*, we mean all of the standard programs developed by the Stanford TeX project directed by Donald E. Knuth: Metafont, DVItype, GFtoDVI, BibTeX, Tangle, etc., as well as TeX itself. Other programs are also included: DVIcopy, written by Peter Breitenlohner, MetaPost and its utilities (derived from Metafont), by John Hobby, etc.

General strategy: Web2c works, as its name implies, by translating the WEB source in which TeX is written into C source code. Its output is not self-contained, however; it makes extensive use of many macros and functions in a library (the `web2c/lib` directory in the sources). Therefore, it will not work without change on an arbitrary WEB program.

Availability: All of Web2c is freely available—"free" both in the sense of no cost (free ice cream) and of having the source code to modify and/or redistribute (free speech). See Section "unixtex.ftp" in *Kpathsea*, for the practical details of how to obtain Web2c.

Different parts of the Web2c distribution have different licensing terms, however, reflecting the different circumstances of their creation; consult each source file for exact details. The main practical implication for redistributors of Web2c is that most of the executables are covered by the GNU General Public License or GNU Lesser General Public License, and therefore anyone who gets a binary distribution must also be able to get the sources, as explained by the terms of the GPL (`https://gnu.org/licenses/`. The GPL covers the Web2c executables, including `tex`, because the Free Software Foundation sponsored the initial development of the Kpathsea library that Web2c uses. The basic source files from Stanford, however, have their own copyright terms or are in the public domain, and are not covered by the GPL.

History: Tomas Rokicki originated the TeX-to-C system in 1987, working from the first change files for TeX under Unix, which were done primarily by Howard Trickey and Pavel Curtis. Tim Morgan then took over development and maintenance for a number of years; the name changed to Web-to-C somewhere in there. In 1990, Karl Berry became the maintainer. He made many changes to the original sources, and started using the shorter name Web2c. In 1997, Olaf Weber took over, and then in 2006, Karl started taking care of it again. No significant development or changes have been needed for many years. Dozens of other people have contributed; their names are listed in the `ChangeLog` files.

Other acknowledgements by Karl: The University of Massachusetts at Boston (particularly Rick Martin and Bob Morris) provided computers and ftp access to me for many years. Richard Stallman at the Free Software Foundation employed me while I wrote the original path searching library (for the GNU font utilities). (rms also gave us Emacs, GDB, and GCC, without which I cannot imagine developing Web2c.) And, of course, TeX would not exist in the first place without Donald E. Knuth.

Further reading: See Appendix B [References], page 55.

2 Installation

(A copy of this chapter is in the distribution file `web2c/INSTALL`.)

Installing Web2c is mostly the same as installing any other Kpathsea-using program. Therefore, for the basic steps involved, see Section “Installation” in *Kpathsea*. (A copy is in the file `kpathsea/INSTALL`.)

One peculiarity to Web2c is that the source distribution comes in two files: `web.tar.gz` and `web2c.tar.gz`. You must retrieve and unpack them both. (We have two because the former archive contains the very large and seldom-changing original WEB source files.) See Section “unixtex.ftp” in *Kpathsea*.

Another peculiarity is the MetaPost program. Although it has been installed previously as `mp`, as of Web2c 7.0 the installed name is now `mpost`, to avoid conflict with the `mp` program that does prettyprinting. This approach was recommended by the MetaPost author, John Hobby. If you as the TEX administrator wish to make it available under its shorter name as well, you will have to set up a link or some such yourself. And of course individual users can do the same.

For solutions to common installation problems and information on how to report a bug, see the file `kpathsea/BUGS` (see Section “Bugs” in *Kpathsea*). See also the Web2c home page, `http://www.tug.org/web2c`.

Points worth repeating:

- Before starting the standard compilation and installation you must install the basic fonts, macros, and other library files. See Section “Installation” in *Kpathsea*.
- If you do not wish to use the standard file locations, see Section “Changing search paths” in *Kpathsea*.
- Some Web2c features are enabled or disabled at `configure` time, as described in the first section below.

2.1 `configure` options

This section gives pointers to descriptions of the ‘`--with`’ and ‘`--enable`’ `configure` arguments that Web2c accepts. Some are specific to Web2c, others are generic to all Kpathsea-using programs.

For a list of all the options `configure` accepts, run ‘`configure --help`’. The generic options are listed first, and the package-specific options come last.

For a description of the generic options (which mainly allow you to specify installation directories) and basic `configure` usage, see Section “Running `configure` scripts” in *Autoconf*, a copy is in the file `kpathsea/CONFIGURE`.

‘`--disable-dump-share`’

Do not make fmt/base/mem files sharable across different endian architectures. See Section 4.3.3 [Hardware and memory dumps], page 12.

'`--without-maketexmf-default`'
'`--without-maketexpk-default`'
'`--without-maketextfm-default`'
'`--with-maketextex-default`'
: Enable or disable the dynamic generation programs. See Section "mktex configuration" in *Kpathsea*. The defaults are the inverse of the options, i.e., everything is enabled except `mktextex`.

'`--enable-shared`'
: Build Kpathsea as a shared library. See Section "Shared library" in *Kpathsea*.

'`--with-editor=`*`cmd`*'
: Change the default editor invoked by the '`e`' interactive command. See Section 4.4 [Editor invocation], page 13.

'`--with-epsfwin`'
'`--with-hp2627win`'
'`--with-mftalkwin`'
'`--with-nextwin`'
'`--with-regiswin`'
'`--with-suntoolswin`'
'`--with-tektronixwin`'
'`--with-unitermwin`'
'`--with-x`'
'`--with-x-toolkit=KIT`'
'`--with-x11win`'
'`--with-x11`'
: Define Metafont graphics support; by default, no graphics support is enabled. See Section 6.4 [Online Metafont graphics], page 29.

'`--x-includes=`*`dir`*'
'`--x-libraries=`*`dir`*'
: Define the locations of the X11 include files and libraries; by default, `configure` does its best to guess). See Section "Optional Features" in *Autoconf*. A copy is in `kpathsea/CONFIGURE`.

2.2 Compile-time options

In addition to the `configure` options listed in the previous section, there are a few things that can be affected at compile-time with C definitions, rather than with `configure`. Using any of these is unusual.

To specify extra compiler flags ('`-D`*`name`*' in this case), the simplest thing to do is:

```
make XCFLAGS="ccoptions"
```

You can also set the `CFLAGS` environment variable before running `configure`. See Section "configure environment" in *Kpathsea*.

Anyway, here are the possibilities:

'`-DFIXPT`'
'`-DNO_MF_ASM`'
: Use the original WEB fixed-point routines for Metafont and MetaPost arithmetic calculations regarding fractions. By default, assembly-language routines are used on x86 hardware with GNU C (unless '`NO_MF_ASM`' is defined), and floating-point routines are used otherwise.

'`-DIPC_DEBUG`'
: Report on various interprocess communication activities. See Section 5.6 [IPC and TeX], page 24.

2.3 Additional targets

Web2c has several Make targets besides the standard ones. You can invoke these either in the top level directory of the source distribution (the one containing `kpathsea/` and `web2c/`), or in the `web2c/` directory.

'`c-sources`'
: Make only the C files, translated from the Web sources, presumably because you want to take them to a non-Unix machine.

'`formats`'
'`install-formats`'
: Make or install all the memory dumps (see Section 4.3 [Memory dumps], page 11). By default, the standard plain formats plus `latex.fmt` are made. You can add other formats by redefining the `fmts`, `bases`, and `mems` variables. See the top of `web2c/Makefile` for the possibilities.

'`fmts`'
'`install-fmts`'
: Make or install the TeX `.fmt` files. See Section 5.2 [Initial TeX], page 18.

'`bases`'
'`install-bases`'
: Make or install the Metafont `.base` files. See Section 6.2 [Initial Metafont], page 27.

'`mems`'
'`install-mems`'
: Make or install the MetaPost `.mem` files. See Section 7.2 [Initial MetaPost], page 35.

'`triptrap`'
'`trip`'
'`trap`'
'`mptrap`'
: To run the torture tests for TeX, Metafont, and MetaPost (respectively). See the next section.

2.4 Trip, trap, and mptrap: Torture tests

To validate your TeX, Metafont, and MetaPost executables, run '`make triptrap`'. This runs the trip, trap, and mptrap "torture tests". See the files `triptrap/tripman.tex`, `triptrap/trapman.tex`, and `triptrap/mptrap.readme` for detailed information and background on the tests.

The differences between your executables' behavior and the standard values will show up on your terminal. The usual differences (these are all acceptable) are:

- string usage and table sizes;
- glue set ratios;
- '`down4`', '`right4`', and '`y4`' commands in DVItype output;
- dates and times.

Any other differences are trouble. The most common culprit in the past has been compiler bugs, especially when optimizing. See Section "TEX or Metafont failing" in *Kpathsea*.

The files `trip.diffs`, `mftrap.diffs`, and `mptrap.diffs` in the `triptrap` directory show the standard diffs against the original output. If you diff your diffs against these files, you should come up clean. For example

```
make trip >&mytrip.diffs
diff triptrap/trip.diffs mytrip.diffs
```

To run the tests separately, use the targets `trip`, `trap`, and `mptrap`.

To run simple tests for all the programs as well as the torture tests, run '`make check`'. You can compare the output to the distributed file `tests/check.log` if you like.

3 Commonalities

Many aspects of the TEX system are the same among more than one program, so we describe those pieces together, here.

3.1 Option conventions

To provide a clean and consistent behavior, we chose to have all these programs use the GNU function `getopt_long_only` to parse command lines. However, we do use in a restricted mode, where all the options have to come before the rest of the arguments.

As a result, you can:

- use '`-`' or '`--`' to start an option name;
- use any unambiguous abbreviation for an option name;
- separate option names and values with either '`=`' or one or more spaces;
- use filenames that would otherwise look like options by putting them after an option '`--`'.

By convention, non-option arguments, if specified, generally define the name of an input file, as documented for each program.

If a particular option with a value is given more than once, it is the last value that counts.

For example, the following command line specifies the options '`foo`', '`bar`', and '`verbose`'; gives the value '`baz`' to the '`abc`' option, and the value '`xyz`' to the '`quux`' option; and specifies the filename `-myfile-`.

```
-foo --bar -verb -abc=baz -quux karl --quux xyz -- -myfile-
```

3.2 Common options

All of these programs accept the standard GNU '`--help`' and '`--version`' options, and several programs accept '`--verbose`'. Rather than writing identical descriptions for every program, they are described here.

'`--help`' Print a usage message listing basic usage and all available options to standard output, then exit successfully.

'`--verbose`'
Print progress reports to standard output.

'`--version`'
Print the version number to standard output, then exit successfully.

TEX, Metafont, and MetaPost have a number of additional options in common:

'`-cnf-line=`*`str`*'
Parse *str* as if it were a line in the `texmf.cnf` configuration file, overriding all other settings. See Section "Path searching options" in *Kpathsea*.

'`-file-line-error`'
'`-no-file-line-error`'
Change (or do not change) the way error messages are printed. The alternate style looks like error messages from many compilers and is easier

to parse for some editors that invoke TEX. This option used to be called '`-file-line-error-style`'.

'`-fmt=`*`dumpname`*'
'`-base=`*`dumpname`*'
'`-mem=`*`dumpname`*'
: Use *dumpname* instead of the program name or a '`%&`' line to determine the name of the memory dump file read ('`fmt`' for TEX, '`base`' for Metafont, '`mem`' for MetaPost). See Section 4.3 [Memory dumps], page 11. Also sets the program name to *dumpname* if no '`-progname`' option was given.

'`-halt-on-error`'
: Stop processing and exit when an error occurs, as opposed to the normal process of trying to recover and continue.

'`-ini`' Enable the "initial" form of the program (see Section 4.2 [Initial and virgin], page 11). This is implicitly set if the program name is `initex` resp. `inimf`.

'`-interaction=`*`string`*'
: Set the interaction mode from the command line. The *string* must be one of '`batchmode`', '`nonstopmode`', '`scrollmode`', or '`errorstopmode`'.

'`-jobname=`*`string`*'
: Set the job name to *string*, instead of deriving it from the name of the input file.

'`-kpathsea-debug=`*`number`*'
: Set path searching debugging flags according to the bits of *number* (see Section "Debugging" in *Kpathsea*). You can also specify this in `KPATHSEA_DEBUG` environment variable (for all Web2c programs). (The command line value overrides.) The most useful value is '`-1`', to get all available output.

'`-output-directory=`*`dirname`*'
: Specify the directory *dirname* to which output files are written. Also look for input files in *dirname* first, before looking along the normal search path. See Section 3.4 [Output file location], page 8.

'`-parse-first-line`'
'`-no-parse-first-line`'
: Check or disable checking whether the first line of the main input file starts with '`%&`', and parse it if it does. This line can be used specify the format and/or a TCX file.

'`-progname=`*`string`*'
: Set program (and memory dump) name to *string*. This may affect the search paths and other values used (see Section "Config files" in *Kpathsea*). Using this option is equivalent to making a link named *string* to the binary and then invoking the binary under that name. See Section 4.3 [Memory dumps], page 11.

'`-recorder`'
: Enable the filename recorder. This makes the program save a list of the opened files into a file with (by default) extension '`.fls`'. For Aleph, this option is always on, and the file has extension '`.ofl`'.

Ordinarily, the '`.fls`' file is written to the same location as the '`.log`' file, for example, respecting `-output-directory` if it is given (see Section 3.4 [Output file location], page 8). However, if TEX processing is done on the command line (or in response to the '`**`' prompt), the '`.fls`' might be written to the current directory, or include an integer (the current pid), as in `texput1234.fls`. You can use `-jobname` to explicitly set the basename.

'`-translate-file=`*`tcxfile`*'
Use *tcxfile* to define which characters are printable and translations between the internal and external character sets. Moreover, *tcxfile* can be explicitly declared in the first line of the main input file '`%& -translate-file=`*`tcxfile`*'. This is the recommended method for portability reasons. See Section 5.4.2 [TCX files], page 21.

'`-8bit`' This option specifies that by default all characters should be considered printable. If '`-translate-file`' was given as well, then the TCX file may mark characters as non-printable. This is a no-op in engines natively supporting Unicode.

3.3 Path searching

All of the Web2c programs, including TEX, which do path searching use the Kpathsea routines to do so. The precise names of the environment and configuration file variables which get searched for particular file formatted are therefore documented in the Kpathsea manual (see Section "Supported file formats" in *Kpathsea*). Reading `texmf.cnf` (see Section "Config files" in *Kpathsea*), invoking `mktex...` scripts (see Section "mktex scripts" in *Kpathsea*), and so on are all handled by Kpathsea.

The programs which read fonts make use of another Kpathsea feature: `texfonts.map`, which allows arbitrary aliases for the actual names of font files; for example, '`Times-Roman`' for '`ptmr8r.tfm`'. The distributed (and installed by default) `texfonts.map` includes aliases for many widely available PostScript fonts by their PostScript names.

3.4 Output file location

All the programs generally follow the usual convention for output files. Namely, they are placed in the directory current when the program is run, regardless of any input file location; or, in a few cases, output is to standard output.

For example, if you run '`tex /tmp/foo`', for example, the output will be in `./foo.dvi` and `./foo.log`, not `/tmp/foo.dvi` and `/tmp/foo.log`.

You can use the '`-output-directory`' option to cause all output files that would normally be written in the current directory to be written in the specified directory instead. See Section 3.2 [Common options], page 6.

If the current directory is not writable, and '`-output-directory`' is not specified, the main programs (TEX, Metafont, MetaPost, and BibTEX) make an exception: if the config file or environment variable value `TEXMFOUTPUT` is set (it is not by default), output files are written to the directory specified.

`TEXMFOUTPUT` is also checked for input files, as TeX often generates files that need to be subsequently read; for input, no suffixes (such as '`.tex`') are added by default and no exhaustive path searching is done, the input name is simply checked as given.

4 Three programs: Metafont, MetaPost, and TeX

TeX, Metafont, and MetaPost have a number of features in common. Besides the ones here, command-line options and other commonalities are described in the previous section.

4.1 Runtime options

Besides the configure- and compile-time options described in the installation section (see Chapter 2 [Installation], page 2), you can control a number of parameters in the `texmf.cnf` runtime file read by Kpathsea (see Section "Config files" in *Kpathsea*).

The main purpose of `texmf.cnf` is to specify search paths, but array sizes and other options are also set there. Most are rather obscure. Here are a few of the more interesting values:

'`main_memory`'
: Total words of memory available, for TeX, Metafont, and MetaPost. Must remake the format file after changing.

'`extra_mem_bot`'
: Extra space for "large" TeX data structures (default 0): boxes, glue, breakpoints, et al. If you use PiCTeX, you may well want to set this.

'`expand_depth`'
: Limit on recursive expansion calls before TeX aborts (default 10000). If a TeX program does an unterminated recursive expansion, TeX will dutifully expand macros until the system's runtime stack overflows, typically with a segmentation fault (SIGSEGV). This parameter was introduced to minimize the chance of that unpleasant (though not dangerous) crash, instead allowing TeX to quit with a more informative message.

  The crash can still happen, though, if the system has an exceptionally small memory allocation for its stack. There is no quantitative way to determine the limit, and it does not seem worth implementing system-dependent heuristics to guess at the number, since it's highly improbable that any real TeX code will ever need more than 10000 recursive expansions (it has never happened). For the same reason, using the libsigsegv library (`https://gnu.org/s/libsigsegv`) does not seem worth the effort.

'`texmf_casefold_search`'
: See Section "Casefolding search" in *Kpathsea*.

Ideally all arrays would be dynamically expanded as necessary, so the only limiting factor would be the amount of swap space available, or some configurable limit much larger than can currently be supported.

Unfortunately, implementing this is extremely difficult, as the fixed size of arrays is assumed in many places throughout the source code. These runtime limits are a practical compromise between the compile-time limits in previous versions, and truly dynamic arrays. (On the other hand, the Web2c BibTeX implementation does do true dynamic reallocation of some arrays.)

Nowadays there is rarely a reason to modify the values. But if you do wish to modify `texmf.cnf`, in TeX Live the best approach is to put your changes, and only your changes at the top of the TL installation tree. That is, if the system `texmf.cnf` is installed in `/some/path/to/texlive/YYYY/texmf-dist/web2c/texmf.cnf` is put your custom settings in `/some/path/to/texlive/YYY/texmf.cnf`, where *YYYY* is the year of installation (if you use that subdirectory; it's the default). That way, unrelated changes to the system `texmf.cnf` can happen with normal updates, without affecting your local values.

4.2 Initial and virgin

The TEX and Metafont programs each have two main variants, called *initial* and *virgin*. MetaPost no longer makes this distinction.

The initial form is enabled if:

1. the '`-ini`' option was specified; or
2. the program name is `initex` resp. `inimf`; or
3. the first line of the main input file is '`%&ini`';

otherwise, the virgin form is used.

The *virgin* form is the one generally invoked for production use. The first thing it does is read a memory dump (see Section 4.3.2 [Determining the memory dump to use], page 12), and then proceeds on with the main job.

The *initial* form is generally used only to create memory dumps (see the next section). It starts up more slowly than the virgin form, because it must do lengthy initializations that are encapsulated in the memory dump file.

4.3 Memory dumps

In typical use, TEX and Metafont require a large number of macros to be predefined; therefore, they support *memory dump* files, which can be read much more efficiently than ordinary source code.

4.3.1 Creating memory dumps

The programs all create memory dumps in slightly idiosyncratic (thought substantially similar) way, so we describe the details in separate sections (references below). The basic idea is to run the initial version of the program (see Section 4.2 [Initial and virgin], page 11), read the source file to define the macros, and then execute the `\dump` primitive.

Also, each program uses a different filename extension for its memory dumps, since although they are completely analogous they are not interchangeable (TEX cannot read a Metafont memory dump, for example).

Here is a list of filename extensions with references to examples of creating memory dumps:

TEX	('`.fmt`') See Section 5.2 [Initial TEX], page 18.
Metafont	('`.base`') See Section 6.2 [Initial Metafont], page 27.

When making memory dumps, the programs read environment variables and configuration files for path searching and other values as usual. If you are making a new installation and have environment variables pointing to an old one, for example, you will probably run into difficulties.

4.3.2 Determining the memory dump to use

The virgin form (see Section 4.2 [Initial and virgin], page 11) of each program always reads a memory dump before processing normal source input. All three programs determine the memory dump to use in the same way:

1. If the first non-option command-line argument begins with '`&`', the program uses the remainder of that argument as the memory dump name. For example, running '`tex \&super`' reads `super.fmt`. (The backslash protects the '`&`' against interpretation by the shell.)
2. If the '`-fmt`' resp. '`-base`' option is specified, its value is used.
3. If the '`-progname`' option is specified, its value is used.
4. If the first line of the main input file (which must be specified on the command line, not in response to '`**`') is `%&`*`dump`*, and *dump* is an existing memory dump of the appropriate type, *dump* is used.

   The first line of the main input file can also specify which character translation file is to be used: `%&-translate-file=`*`tcxfile`* (see Section 5.4.2 [TCX files], page 21).

   These two roles can be combined: `%&`*`dump`* `-translate-file=`*`tcxfile`*. If this is done, the name of the dump must be given first.
5. Otherwise, the program uses the program invocation name, most commonly `tex` resp. `mf`. For example, if `latex` is a link to `tex`, and the user runs '`latex foo`', `latex.fmt` will be used.

4.3.3 Hardware and memory dumps

By default, memory dump files are generally sharable between architectures of different types; specifically, on machines of different endianness (see Section "Byte order" in *GNU C Library*). (This is a feature of the Web2c implementation, and is not true of all TEX implementations.) If you specify '`--disable-dump-share`' to `configure`, however, memory dumps will be endian-dependent.

The reason to do this is speed. To achieve endian-independence, the reading of memory dumps on LittleEndian architectures, such as PC's and DEC architectures, is somewhat slowed (all the multibyte values have to be swapped). Usually, this is not noticeable, and the advantage of being able to share memory dumps across all platforms at a site far outweighs the speed loss. But if you're installing Web2c for use on LittleEndian machines only, perhaps on a PC being used only by you, you may wish to get maximum speed.

TEXnically, even without '`--disable-dump-share`', sharing of `.fmt` files cannot be guaranteed to work. Floating-point values are always written in native format, and hence will generally not be readable across platforms. Fortunately, TEX uses floating point only to represent glue ratios, and all common formats (plain, LATEX, AMSTEX, ...) do not do any glue setting at `.fmt`-creation time. Metafont does not use floating point in any dumped value at all.

Incidentally, different memory dump files will never compare equal byte-for-byte, because the program always dumps the current date and time. So don't be alarmed by just a few bytes difference.

If you don't know what endianness your machine is, and you're curious, here is a little C program to tell you. (The `configure` script contains a similar program.) This is from the book *C: A Reference Manual*, by Samuel P. Harbison and Guy L. Steele Jr. (see Appendix B [References], page 55).

```
main ()
{
  /* Are we little or big endian?  From Harbison&Steele.  */
  union
  {
    long l;
    char c[sizeof (long)];
  } u;
  u.l = 1;
  if (u.c[0] == 1)
    printf ("LittleEndian\n");
  else if (u.c[sizeof (long) - 1] == 1)
    printf ("BigEndian\n");
  else
    printf ("unknownEndian");

  exit (u.c[sizeof (long) - 1] == 1);
}
```

4.4 Editor invocation

TEX, Metafont, and MetaPost all (by default) stop and ask for user intervention at an error. If the input came from a file, and the user responds with *e* or *E*, the program invokes an editor.

Specifying '`--with-editor=`*cmd*' to `configure` sets the default editor command string to *cmd*. The environment variables/configuration values `TEXEDIT`, `MFEDIT`, and `MPEDIT` (respectively) override this. If '`--with-editor`' is not specified, the default is `vi +%d %s` on Unix, and an invocation of the TEXworks editor on Windows. (See `texmf.cnf` for the precise values.)

In this string, '`%d`' is replaced by the line number of the error, and '`%s`' is replaced by the name of the current input file.

4.5 `\input` filenames

TEX, Metafont, and MetaPost source programs can all read other source files with the `\input` (TEX) and `input` (MF and MP) primitives:

```
\input name % in TeX
```

The file *name* can always be terminated with whitespace; for Metafont and MetaPost, the statement terminator '`;`' also works. (LATEX and other macro packages provide other interfaces to `\input` that allow different notation; here we are concerned only with the primitive operation.)

As (allowed) extensions to standard TEX, Web2c also supports specifying the filename in double quotes (`"some name"`) and in braces (`{"some name"}`), which is convenient for filenames containing spaces or other special characters, as described in the sections below.

In all cases, space tokens are ignored after the filename is read.

Also, double quote (`"`) characters are ignored within the filename; there is no way to read files whose names contain a `"`.

However, for maximal portability of your document across systems, use only the characters '`a`'–'`z`', '`0`'–'`9`', and at most one '`.`'. Do not use anything but simple filenames, since directory separators vary among systems; instead, add the necessary directories to the appropriate search path.

4.5.1 `\input` quoted filename: `\input "some name"`

As of Web2c version 7.5.3 (2004), double-quote characters can be used to include spaces or other special characters. In typical use, the '`"`' characters surround the entire filename:

```
\input "filename with spaces"
```

Technically, the quote characters can be used inside the name, and can enclose any characters, as in:

```
\input filename" "with" "spaces
```

One more point. In LaTeX, the quotes are needed inside the braces of its `\input` macro, thus:

```
\input{a b}    % fails
\input{"a b"}  % ok
```

There is no way to quote the quote character.

4.5.2 `\input` braced filename: `\input{some name}`

As of Web2c 2020, `\input` filenames in TeX engines (this does not apply in Metafont and MetaPost) can also be specified within a TeX group, typically curly braces. For example:

```
\input{filename with spaces}
```

As always with TeX, the brace characters are not hardwired; what counts is the category code: the first token after the `\input` must be of catcode 1 (begin group), and it is matched with the next character of catcode 2 (end group).

Within the group-delimited filename, braces are treated as normal characters.

As with all forms of filenames, following spaces are ignored (after the end group), and double quote (`"`) characters are ignored within the filename.

4.5.3 `\input` filename caveats

The quoting mechanisms just described come into play *after* TeX has tokenized and expanded the input. So, multiple spaces and tabs will generally be seen as a single space, active characters such as '`~`' are expanded first (generally causing an error), and so on. More examples below.

On the other hand, various C library routines and Unix itself use the null byte (character code zero, ASCII NUL) to terminate strings. So filenames in Web2c cannot contain nulls, even though TeX itself does not treat NUL specially.

Finally, the present Web2c implementation does '`~`' and '`$`' expansion on *name*, unlike Knuth's original implementation. Thus:

```
\input ~jsmith/$foo.bar
```

will dereference the environment variable or Kpathsea config file value '`foo`' and read that file, extended with '`.bar`', in user '`jsmith`''s home directory. You can also use braces in the variable expansion, as in '`${foo}bar`', if you want to follow the variable name with a letter, numeral, or '`_`'.

(So another way to get a program to read a filename containing whitespace is to define an environment variable and dereference it.)

In all the common TEX formats (plain TEX, LATEX, ConTEXt, AMSTEX, ...), the characters '`~`' and '`$`' have special category codes, so to actually use these in a document you have to change their catcodes or use `\string`.

5 TEX: Typesetting

TEX is a typesetting system: it was especially designed to handle complex mathematics, as well as most ordinary text typesetting.

TEX is a batch language, like C or Pascal, and not an interactive "word processor": you compile a TEX input file into a corresponding device-independent (DVI) file (and then translate the DVI file to the commands for a particular output device). This approach has both considerable disadvantages and considerable advantages. For a complete description of the TEX language, see *The TEXbook* (see Appendix B [References], page 55). Many other books on TEX, introductory and otherwise, are available.

5.1 `tex` invocation

TEX (usually invoked as `tex`) formats the given text and commands, and outputs a corresponding device-independent representation of the typeset document. This section merely describes the options available in the Web2c implementation. For a complete description of the TEX typesetting language, see *The TEXbook* (see Appendix B [References], page 55).

TEX, Metafont, and MetaPost process the command line (described here) and determine their memory dump (fmt) file in the same way (see Section 4.3 [Memory dumps], page 11). Synopses:

```
tex [option]... [texname[.tex]] [tex-commands]
tex [option]... \first-line
tex [option]... &fmt args
```

TEX searches the usual places for the main input file *texname* (see Section "Supported file formats" in *Kpathsea*), extending *texname* with `.tex` if necessary. To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

After *texname* is read, TEX processes any remaining *tex-commands* on the command line as regular TEX input. Also, if the first non-option argument begins with a TEX escape character (usually `\`), TEX processes all non-option command-line arguments as a line of regular TEX input.

If no arguments or options are specified, TEX prompts for an input file name with '`**`'.

TEX writes the main DVI output to the file *`basetexname`*`.dvi`, where *basetexname* is the basename of *texname*, or '`texput`' if no input file was specified. A DVI file is a device-independent binary representation of your TEX document. The idea is that after running TEX, you translate the DVI file using a separate program to the commands for a particular output device, such as a PostScript printer (see Section "Introduction" in *Dvips*) or an X Window System display (see xdvi(1)).

TEX also reads TFM files for any fonts you load in your document with the `\font` primitive. By default, it runs an external program named `mktextfm` to create any nonexistent TFM files. You can disable this at configure-time or runtime (see Section "mktex configuration" in *Kpathsea*). This is enabled mostly for the sake of the EC fonts, which can be generated at any size.

TEX can write output files, via the `\openout` primitive; this opens a security hole vulnerable to Trojan horse attack: an unwitting user could run a TEX program that overwrites, say, `~/.rhosts`. (MetaPost has a `write` primitive with similar implications). To alleviate this and similar problems the functions `kpathsea_out_name_ok` and `kpathsea_in_name_ok` from the Kpathse library (see Section "Calling sequence" in *Kpathsea*) are used to determine if a given filename is acceptable to be opened for output or input, depending on the setting of the configuration variables `openout_any` and `openin_any`: '`a`' (for "any", the default for `openin_any`), '`r`' (for "restricted"), or '`p`' (for "paranoid", the default for `openout_any`).

In any case, all `\openout` filenames are recorded in the log file, except those opened on the first line of input, which is processed when the log file has not yet been opened.

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-enc`'
'`-[no]-file-line-error`'
'`-fmt=`*`fmtname`*'
'`-halt-on-error`'
'`-ini`'
'`-interaction=`*`string`*'
'`-ipc`'
'`-ipc-start`'
'`-jobname=`*`string`*'
'`-kpathsea-debug=`*`number`*'
'`-[no]parse-first-line`'
'`-output-directory`'
'`-progname=`*`string`*'
'`-recorder`'
'`-translate-file=`*`tcxfile`*'
'`-8bit`' These options are common to TEX, Metafont, and MetaPost. See Section 3.2 [Common options], page 6.

'`-enc`' Enable encTEX extensions, such as `\mubyte`. This can be used to support the Unicode UTF-8 input encoding, although using an engine with native Unicode support is more common nowadays. `http://www.olsak.net/enctex.html`.

'`-ipc`'
'`-ipc-start`'
With either option, TEX writes its DVI output to a socket as well as to the usual `.dvi` file. With '`-ipc-start`', TEX also opens a server program at the other end to read the output. See Section 5.6 [IPC and TEX], page 24.

These options are available only if the '`--enable-ipc`' option was specified to `configure` during installation of Web2c.

'`-mktex=`*`filetype`*'
'`-no-mktex=`*`filetype`*'
Turn on or off the '`mktex`' script associated with *filetype*. For TEX proper, *filetype* can only be '`tex`' and '`tfm`', but for pdfTEX and luaTEX, it can also be '`pk`'.

'`-mltex`' If we are `INITEX` (see Section 4.2 [Initial and virgin], page 11), enable MLTEX extensions such as `\charsubdef`. Implicitly set if the program name is `mltex`. See Section 5.4.1 [MLTEX], page 20.

'`-output-comment=`*`string`*'
: Use *string* as the DVI file comment. Ordinarily, this comment records the date and time of the TEX run, but if you are doing regression testing, you may not want the DVI file to have this spurious difference. This is also taken from the environment variable and config file value '`output_comment`'.

'`-shell-escape`'
'`-no-shell-escape`'
'`-shell-restricted`'
: Enable, or disable, or enable with restrictions the `\write18{`*`shell-command`*`}` feature for external executing shell commands. See Section 5.5 [Shell escapes], page 23.

'`-enable-write18`'
'`-disable-write18`'
: Synonyms for `-shell-escape` and `-no-shell-escape`, for compatibility with MiKTEX. (MiKTEX also accepts both pairs of options.) See Section 5.5 [Shell escapes], page 23.

'`-src-specials`'
'`-src-specials=`*`string`*'
: This option makes TEX output specific source information using '`\special`' commands in the DVI file. These '`\special`' track the current file name and line number.

Using the first form of this option, the '`\special`' commands are inserted automatically.

In the second form of the option, *string* is a comma separated list of the following values: '`cr`', '`display`', '`hbox`', '`math`', '`par`', '`parend`', '`vbox`'. You can use this list to specify where you want TEX to output such commands. For example, '`-src-specials=cr,math`' will output source information every line and every math formula.

These commands can be used with the appropriate DVI viewer and text editor to switch from the current position in the editor to the same position in the viewer and back from the viewer to the editor.

This option works by inserting '`\special`' commands into the token stream, and thus in principle these additional tokens can be recovered or seen by the tricky-enough macros. If you run across a case, let us know, because this counts as a bug. However, such bugs are very hard to fix, requiring significant changes to TEX, so please don't count on it.

Redefining '`\special`' will not affect the functioning of this option. The commands inserted into the token stream are hard-coded to always use the '`\special`' primitive.

TEX does not pass the trip test when this option is enabled.

5.2 Initial TEX

The *initial* form of TEX is invoked by '`tex -ini`'. It does lengthy initializations avoided by the "virgin" (`vir`) form, so as to be capable of dumping '`.fmt`' files (see Section 4.3

[Memory dumps], page 11). For a detailed comparison of virgin and initial forms, see Section 4.2 [Initial and virgin], page 11.

For a list of options and other information, see Section 5.1 [tex invocation], page 16.

Unlike Metafont and MetaPost, many format files are commonly used with TEX. The standard one implementing the features described in the *TEXbook* is '`plain.fmt`', also known as '`tex.fmt`' (again, see Section 4.3 [Memory dumps], page 11). It is created by default during installation, but you can also do so by hand if necessary (e.g., if an update to `plain.tex` is issued):

```
tex -ini '\input plain \dump'
```

(The quotes prevent interpretation of the backslashes from the shell.) Then install the resulting `plain.fmt` in '`$(fmtdir)`' (`/usr/local/share/texmf/web2c` by default), and link `tex.fmt` to it.

The necessary invocation for generating a format file differs for each format, so instructions that come with the format should explain. The top-level `web2c` Makefile has targets for making most common formats: `plain latex amstex texinfo eplain`. See Section 5.3 [Formats], page 19, for more details on TEX formats.

5.3 Formats

TEX *formats* are large collections of macros, often dumped into a `.fmt` file (see Section 4.3 [Memory dumps], page 11) by `tex -ini` (see Section 5.2 [Initial TeX], page 18). A number of formats are in reasonably widespread use, and the Web2c Makefile has targets to make the versions current at the time of release. You can change which formats are automatically built by setting the `fmts` Make variable; by default, only the '`plain`' and '`latex`' formats are made.

Nowadays, the formats are generally installed and updated as part of a larger TEX distribution, such as TEX Live (`https://tug.org/texlive`).

`latex` The most widely used format. The current release is named 'LATEX2e'; new versions are released approximately every six months, with patches issued as needed. The old release was called 'LATEX 2.09', and is no longer maintained or supported. LATEX attempts to provide generic markup instructions, such as "emphasize", instead of specific typesetting instructions, such as "use the 10 pt Computer Modern italic font". The LATEX home page: `https://www.latex-project.org`.

`context` ConTEXt is an independent macro package which has a basic document structuring approach similar to LATEX. It also supports creating interactive PDF files and has integrated MetaPost support, among many other interesting features. The ConTEXt home page: `http://www.pragma-ade.com`.

`amstex` The official typesetting system of the American Mathematical Society. Like LATEX, it encourages generic markup commands. The AMS also provides many LATEX packages for authors who prefer LATEX. Taken together, they are used to produce nearly all AMS publications, e.g., *Mathematical Reviews*. The AMSTEX home page: `https://www.ams.org/tex`.

`texinfo` The documentation system developed and maintained by the Free Software Foundation for their software manuals. It can be automatically converted into plain text, a machine-readable on-line format called 'info', HTML, etc. The Texinfo home page: `https://www.gnu.org/software/texinfo`.

`eplain` The "expanded plain" format provides various common features (e.g., symbolic cross-referencing, tables of contents, indexing, citations using BibTEX), for those authors who prefer to handle their own high-level formatting. The Eplain home page: `https://tug.org/eplain`.

`slitex` An obsolete LATEX 2.09 format for making slides. It is replaced by the '`slides`' document class, although the '`beamer`' package is the most commonly method for making slides nowadays. The Beamer page on CTAN: `https://ctan.org/pkg/beamer`.

5.4 Languages and hyphenation

TEX supports most natural languages. See also Section 5.7 [TEX extensions], page 24.

5.4.1 MLTEX: Multi-lingual TEX

Multi-lingual TEX (`mltex`) is an extension of TEX originally written by Michael Ferguson and now updated and maintained by Bernd Raichle. With the advent of Unicode, it has become obsolete, though it is still supported in Web2c in the event of bugs or compilation bugs.

MLTEX allows the use of non-existing glyphs in a font by declaring glyph substitutions. These are restricted to substitutions of an accented character glyph, which need not be defined in the current font, by its appropriate `\accent` construction using a base and accent character glyph, which do have to exist in the current font. This substitution is automatically done behind the scenes, if necessary, and thus MLTEX additionally supports hyphenation of words containing an accented character glyph for fonts missing this glyph (e.g., Computer Modern). Standard TEX suppresses hyphenation in this case.

MLTEX works at `.fmt`-creation time: the basic idea is to specify the '`-mltex`' option to TEX when you `\dump` a format. Then, when you subsequently invoke TEX and read that `.fmt` file, the MLTEX features described below will be enabled.

Generally, you use special macro files to create an MLTEX `.fmt` file.

The sections below describe the two new primitives that MLTEX defines. Aside from these, MLTEX is completely compatible with standard TEX.

5.4.1.1 `\charsubdef`: Character substitutions

The most important primitive MLTEX adds is `\charsubdef`, used in a way reminiscent of `\chardef`:

```
\charsubdef composite [=] accent base
```

Each of *composite*, *accent*, and *base* are font glyph numbers, expressed in the usual TEX syntax: `` `\e `` symbolically, `'145` for octal, `"65` for hex, `101` for decimal.

MLTEX's `\charsubdef` declares how to construct an accented character glyph (not necessarily existing in the current font) using two character glyphs (that do exist). Thus it defines whether a character glyph code, either typed as a single character or using the `\char` primitive, will be mapped to a font glyph or to an `\accent` glyph construction.

For example, if you assume glyph code 138 (decimal) for an e-circumflex (ê) and you are using the Computer Modern fonts, which have the circumflex accent in position 18 and lowercase 'e' in the usual ASCII position 101 decimal, you would use `\charsubdef` as follows:

```
\charsubdef 138 = 18 101
```

For the plain TEX format to make use of this substitution, you have to redefine the circumflex accent macro `\^` in such a way that if its argument is character 'e' the expansion `\char138` is used instead of `\accent18 e`. Similar `\charsubdef` declaration and macro redefinitions have to be done for all other accented characters.

To disable a previous `\charsubdef` *c*, redefine *c* as a pair of zeros. For example:

```
\charsubdef '321 = 0 0  % disable N tilde
```

(Octal `'321` is the ISO Latin-1 value for the Spanish N tilde.)

`\charsubdef` commands should only be given once. Although in principle you can use `\charsubdef` at any time, the result is unspecified. If `\charsubdef` declarations are changed, usually either incorrect character dimensions will be used or MLTEX will output missing character warnings. (The substitution of a `\charsubdef` is used by TEX when appending the character node to the current horizontal list, to compute the width of a horizontal box when the box gets packed, and when building the `\accent` construction at `\shipout`-time. In summary, the substitution is accessed often, so changing it is not desirable, nor generally useful.)

5.4.1.2 `\tracingcharsubdef`: Substitution diagnostics

To help diagnose problems with '`\charsubdef`', MLTEX provides a new primitive parameter, `\tracingcharsubdef`. If positive, every use of `\charsubdef` will be reported. This can help track down when a character is redefined.

In addition, if the TEX parameter `\tracinglostchars` is 100 or more, the character substitutions actually performed at `\shipout`-time will be recorded.

5.4.2 TCX files: Character translations

TCX (TEX character translation) files help TEX support direct input of 8-bit international characters if fonts containing those characters are being used. Specifically, they map an input (keyboard) character code to the internal TEX character code (a superset of ASCII).

Of the various proposals for handling more than one input encoding, TCX files were chosen because they follow Knuth's original ideas for the use of the '`xchr`' and '`xord`' tables. He ventured that these would be changed in the WEB source in order to adjust the actual version to a given environment. It turns out, however, that recompiling the WEB sources is not as simple a task as Knuth may have imagined; therefore, TCX files, providing the possibility of changing of the conversion tables on on-the-fly, have been implemented instead.

This approach limits the portability of TEX documents, as some implementations do not support it (or use a different method for input-internal reencoding). It may also be problematic to determine the encoding to use for a TEX document of unknown provenance; in the worst case, failure to do so correctly may result in subtle errors in the typeset output. But we feel the benefits outweigh these disadvantages.

This is entirely independent of the MLTEX extension (see Section 5.4.1 [MLTeX], page 20): whereas a TCX file defines how an input keyboard character is mapped to TEX's internal code, MLTEX defines substitutions for a non-existing character glyph in a font with a `\accent` construction made out of two separate character glyphs. TCX files involve no new primitives; it is not possible to specify that an input (keyboard) character maps to more than one character.

Information on specifying TCX files:

- The best way to specify a TCX file is to list it explicitly in the first line of the main document:

  ```
  %& -translate-file=tcxfile
  ```
- You can also specify a TCX file to be used on a particular TEX run with the command-line option '`-translate-file=`*tcxfile*'.
- TCX files are searched for along the `WEB2C` path.
- Initial TEX (see Section 5.2 [Initial TEX], page 18) ignores TCX files.

The Web2c distribution comes with a number of TCX files. Two important ones are `il1-t1.tcx` and `il2-t1.tcx`, which support ISO Latin 1 and ISO Latin 2, respectively, with Cork-encoded fonts (a.k.a. the LATEX T1 encoding). TCX files for Czech, Polish, and Slovak are also provided.

One other notable TCX file is `empty.tcx`, which is, well, empty. Its purpose is to reset Web2C's behavior to the default (only visible ASCII being printable, as described below) when a format was dumped with another TCX being active—which is in fact the case for everything but plain TEX in the TeX Live and other distributions. Thus:

```
latex somefile8.tex
⇒ terminal etc. output with 8-bit chars
latex --translate-file=empty.tcx somefile8.tex
⇒ terminal etc. output with ^^ notation
```

Syntax of TCX files:

1. Line-oriented. Blank lines are ignored.
2. Whitespace is ignored except as a separator.
3. Comments start with '`%`' and continue to the end of the line.
4. Otherwise, a line consists of one or two character codes, optionally followed by 0 or 1. The last number indicates whether *dest* is considered printable.

   ```
   src [dest [prnt]]
   ```
5. Each character code may be specified in octal with a leading '`0`', hexadecimal with a leading '`0x`', or decimal otherwise. Values must be between 0 and 255, inclusive (decimal).
6. If the *dest* code is not specified, it is taken to be the same as *src*.
7. If the same *src* code is specified more than once, it is the last definition that counts.

Finally, here's what happens: when TEX sees an input character with code *src*, it 1) changes *src* to *dest*; and 2) makes the *dest* code "printable", i.e., printed as-is in diagnostics and the log file rather than in '`^^`' notation.

By default, no characters are translated, and character codes between 32 and 126 inclusive (decimal) are printable.

Specifying translations for the printable ASCII characters (codes 32–127) will yield unpredictable results. Additionally you shouldn't make the following characters printable: `^^I` (TAB), `^^J` (line feed), `^^M` (carriage return), and `^^?` (delete), since TEX uses them in various ways.

Thus, the idea is to specify the input (keyboard) character code for *src*, and the output (font) character code for *dest*.

By default, only the printable ASCII characters are considered printable by TEX. If you specify the '`-8bit`' option, all characters are considered printable by default. If you specify both the '`-8bit`' option and a TCX file, then the TCX can set specific characters to be non-printable.

Both the specified TCX encoding and whether characters are printable are saved in the dump files (like `tex.fmt`). So by giving these options in combination with '`-ini`', you control the defaults seen by anyone who uses the resulting dump file.

When loading a dump, if the '`-8bit`' option was given, then all characters become printable by default.

When loading a dump, if a TCX file was specified, then the TCX data from the dump is ignored and the data from the file used instead.

5.4.3 Patgen: Creating hyphenation patterns

Patgen creates hyphenation patterns from dictionary files for use with TEX. Synopsis:

```
patgen dictionary patterns output translate
```

Each argument is a filename. No path searching is done. The output is written to the file *output*.

In addition, Patgen prompts interactively for other values.

For more information, see *Word hy-phen-a-tion by com-puter* by Frank Liang (see Appendix B [References], page 55), and also the `patgen.web` source file.

The only options are '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6).

5.5 Shell escapes

TEX can execute *shell escapes*, that is, arbitrary shell commands. Although tremendously useful, this also has obvious security implications. Therefore, as of TEX Live 2009, a *restricted* mode for shell escapes is the default mode of operation, which allows executing only certain commands, as specified in the `texmf.cnf` configuration file.

- Unrestricted shell escapes are allowed if the option `--shell-escape` is specified, or if the environment variable or config file value `shell_escape` is set to '`t`' or '`y`' and '`1`'.
- Restricted shell escapes are allowed if `shell_escape` is set to '`p`'. This is the default.
- Shell escapes are completely disabled if `--no-shell-escape` is specified, or if `shell_escape` is set to anything else.

When enabled, the TEX construct to execute a system command is `\write18{`*`shell-command`*`}`; for example:

```
\write18{echo "hello, world"}
```

From TEX's point of view, this is a normal `\write` command, and is therefore subject to the usual TEX expansions. Also, the system call either happens during the '`\output`' routine or right away, according to the absence or presence of the `\immediate` prefix, as usual for `\write`.

The *shell-command* string is passed to the command shell (via the C library function `system`). The output of *shell-command* is not diverted anywhere, so it will not appear in the log file, or anywhere but the terminal output. The exit status of the system call is also not available to TEX.

In unrestricted mode, the argument is simply passed straight to `system` unaltered.

In restricted mode, ASCII double quote characters (`"`) should always be used in the argument to `\write18` where quoting of arguments is needed, as in the example above. This is to achieve some measure of system independence. On Unix systems, these are replaced with single quote (`'`) characters to avoid insecure further expansion. Care is also taken on Windows to avoid additional expansions (from, e.g., `'...'`). Mismatched quotation marks in the command string result in a diagnostic message in the log file; no execution is performed.

After quotation processing, if the first word (delimited by a space or tab) of the command is in the list specified by the `shell_escape_commands` configuration value, the command is executed. Otherwise it is not. In any case, a message is written to the log file.

The `shell_escape_commands` value is a comma-separated list of words. Whitespace is significant, and typically should not be present. The default definition in `texmf.cnf` looks like this, but with more commands included:

```
shell_escape_commands = bibtex,kpsewhich,repstopdf,...
```

pdfTEX and luaTEX support reading (via `\input` and `\openin`) and writing (via `\openout`) from pipes if the first character is '`|`'. The following command is then treated exactly the same as the argument to `\write18`. In these engines, the primitive variable `\pdfshellescape` is set to 0 if shell escapes are disabled, 1 if they are enabled, and 2 if they are enabled with restrictions.

The purpose of this feature is to make it possible for TEX documents to perform useful external actions in the common case of an individual user running a known document on his or her own machine. In such environments as CGI scripts or wikis where the input has to be considered untrustworthy, shell escapes should be completely disabled.

5.6 IPC and TEX

(If anyone uses this feature and needs documentation, write `tex-k@tug.org`.)

This functionality is available only if the '`--enable-ipc`' option was specified to `configure` during installation of Web2c (see Chapter 2 [Installation], page 2).

If you define `IPC_DEBUG` before compilation (e.g., with '`make XCFLAGS=-DIPC_DEBUG`'), TEX will print messages to standard error about its socket operations. This may be helpful if you are, well, debugging.

5.7 Extended TEX engines

The base TEX program has been extended in many ways. Here's a partial list.

e-TeX	Adds many new primitives, including right-to-left typesetting and more registers. Now frozen. More info: `https://ctan.org/pkg/etex`.
Aleph	This adds Unicode support, right-to-left typesetting, and more. Omega was the original program. Aleph is an updated version with a variety of bug fixes, and includes e-TeX. Aleph is not actively maintained. More info: `https://ctan.org/pkg/aleph`, `https://ctan.org/pkg/omega`.
pdfTeX	Can produce PDF as well as DVI files. It also incorporates the e-TeX extensions, new primitives for hypertext and micro-typography, reading/writing from pipes, and much more. In TeX Live, the command `etex` invokes pdfTeX to make all these additions available with DVI output. Home page: `http://pdftex.org`.
LuaTeX	Embeds the Lua programming language (`http://lua.org`) and opens up the TeX typesetting engine to control from Lua, starting from the pdfTeX capabilities as a base. Also natively supports UTF-8 input, the OpenType and TrueType font formats, and use of system fonts. Home page: `http://luatex.org`.
XeTeX	Combines support for Unicode input, the OpenType and TrueType font formats, and use of system fonts with the capabilities of pdfTeX, with the exception of the font expansion part of micro-typography. Home page: `https://tug.org/xetex`.
pTeX upTeX	With additional support for Japanese; pTeX was the original engine, and upTeX has native Unicode support and thus is more useful for Chinese and Korean. More info: `https://ctan.org/pkg/ptex`, `https://ctan.org/pkg/uptex`.
epTeX eupTeX	Further extends pTeX and upTeX with the e-TeX extensions. More info: `https://ctan.org/pkg/eptex`, `https://ctan.org/pkg/euptex`.

6 Metafont: Creating typeface families

Metafont is a system for producing shapes; it was designed for producing complete typeface families, but it can also produce geometric designs, dingbats, etc. And it has considerable mathematical and equation-solving capabilities which can be useful entirely on their own.

Metafont is a batch language, like C or Pascal: you compile a Metafont program into a corresponding font, rather than interactively drawing lines or curves. This approach has both considerable disadvantages (people unfamiliar with conventional programming languages will be unlikely to find it usable) and considerable advantages (you can make your design intentions specific and parameterizable). For a complete description of the Metafont language, see *The METAFONTbook* (see Appendix B [References], page 55).

6.1 `mf` invocation

Metafont (usually invoked as `mf`) reads character definitions specified in the Metafont programming language, and outputs the corresponding font. This section merely describes the options available in the Web2c implementation. For a complete description of the Metafont language, see *The Metafontbook* (see Appendix B [References], page 55).

Metafont processes its command line and determines its memory dump (base) file in a way exactly analogous to MetaPost and TEX (see Section 5.1 [tex invocation], page 16, and see Section 4.3 [Memory dumps], page 11). Synopses:

```
mf [option]... [mfname[.mf]] [mf-commands]
mf [option]... \first-line
mf [option]... &base args
```

Most commonly, a Metafont invocation looks like this:

```
mf '\mode:=mode; mag:=magnification; input mfname'
```

(The single quotes avoid unwanted interpretation by the shell.)

Metafont searches the usual places for the main input file *mfname* (see Section "Supported file formats" in *Kpathsea*), extending *mfname* with `.mf` if necessary. To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program. By default, Metafont runs an external program named `mktexmf` to create any nonexistent Metafont source files you input. You can disable this at configure-time or runtime (see Section "mktex configuration" in *Kpathsea*). This is mostly for the sake of the EC fonts, which can be generated at any size.

Metafont writes the main GF output to the file `basemfname.nnngf`, where *nnn* is the font resolution in pixels per inch, and *basemfname* is the basename of *mfname*, or '`mfput`' if no input file was specified. A GF file contains bitmaps of the actual character shapes. Usually GF files are converted immediately to PK files with GFtoPK (see Section 11.2 [gftopk invocation], page 46), since PK files contain equivalent information, but are more compact. (Metafont output in GF format rather than PK for only historical reasons.)

Metafont also usually writes a metric file in TFM format to `basemfname.tfm`. A TFM file contains character dimensions, kerns, and ligatures, and spacing parameters. TEX reads only this `.tfm` file, not the GF file.

The *mode* in the example command above is a name referring to a device definition (see Section 6.3 [Modes], page 28); for example, `localfont` or `ljfour`. These device definitions must generally be precompiled into the base file. If you leave this out, the default is `proof` mode, as stated in *The Metafontbook*, in which Metafont outputs at a resolution of 2602 dpi; this is usually not what you want. The remedy is simply to assign a different mode—`localfont`, for example.

The *magnification* assignment in the example command above is a magnification factor; for example, if the device is 600 dpi and you specify `mag:=2`, Metafont will produce output at 1200 dpi. Very often, the *magnification* is an expression such as `magstep(.5)`, corresponding to a TeX "magstep", which are factors of $1.2\sqrt{2}$.

After running Metafont, you can use the font in a TeX document as usual. For example:

```
\font\myfont = newfont
\myfont Now I am typesetting in my new font (minimum hamburgers).
```

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-[no]-file-line-error`'
'`-fmt=`*`fmtname`*'
'`-halt-on-error`'
'`-ini`'
'`-interaction=`*`string`*'
'`-jobname=`*`string`*'
'`-kpathsea-debug=`*`number`*'
'`-[no]parse-first-line`'
'`-output-directory`'
'`-progname=`*`string`*'
'`-recorder`'
'`-translate-file=`*`tcxfile`*'
'`-8bit`' These options are common to TeX, Metafont, and MetaPost. See Section 3.2 [Common options], page 6.

'`-mktex=`*`filetype`*'
'`-no-mktex=`*`filetype`*'
Turn on or off the '`mktex`' script associated with *filetype*. The only value that makes sense for *filetype* is '`mf`'.

6.2 Initial Metafont

`inimf` is the "initial" form of Metafont, which does lengthy initializations avoided by the "virgin" (`vir`) form, so as to be capable of dumping '`.base`' files (see Section 4.3 [Memory dumps], page 11). For a detailed comparison of virgin and initial forms, see Section 4.2 [Initial and virgin], page 11.

For a list of options and other information, see Section 6.1 [mf invocation], page 26.

The only memory dump file commonly used with Metafont is the default '`plain.base`', also known as '`mf.base`' (again, see Section 4.3 [Memory dumps], page 11). It is created by default during installation, but you can also do so by hand if necessary (e.g., if a Metafont update is issued):

```
mf -ini '\input plain; input modes; dump'
```

(The quotes prevent interpretation of the backslashes from the shell.) Then install the resulting `plain.base` in '`$(basedir)`' (`/usr/local/share/texmf/web2c` by default), and link `mf.base` to it.

For an explanation of the additional `modes.mf` file, see Section 6.3 [Modes], page 28. This file has no counterpart in TEX or MetaPost.

In the past, it was sometimes useful to create a base file `cmmf.base` (a.k.a. `cm.base`), with the Computer Modern macros also included in the base file. Nowadays, however, the additional time required to read `cmbase.mf` is exceedingly small, usually not enough to be worth the administrative hassle of updating the `cmmf.base` file when you install a new version of `modes.mf`. People actually working on a typeface may still find it worthwhile to create their own base file, of course.

6.3 Modes: Device definitions for Metafont

Running Metafont and creating Metafont base files requires information that TEX and MetaPost do not: *mode* definitions which specify device characteristics, so Metafont can properly rasterize the shapes.

When making a base file, a file containing modes for locally-available devices should be input after `plain.mf`. One commonly used file is `ftp://ftp.tug.org/tex/modes.mf`; it includes all known definitions.

If, however, for some reason you have decreased the memory available in your Metafont, you may need to copy `modes.mf` and remove the definitions irrelevant to you (probably most of them) instead of using it directly. (Or, if you're a Metafont hacker, maybe you can suggest a way to redefine `mode_def` and/or `mode_setup`; right now, the amount of memory used is approximately four times the total length of the `mode_def` names, and that's a lot.)

If you have a device not included in `modes.mf`, please see comments in that file for how to create the new definition, and please send the definition to `tex-fonts@math.utah.edu` to get it included in the next release of `modes.mf`.

Usually, when you run Metafont you must supply the name of a mode that was dumped in the base file. But you can also define the mode characteristics dynamically, by invoking Metafont with an assignment to `smode` instead of `mode`, like this:

```
mf '\smode:="newmode.mf"; mag:=magnification; input mfname'
```

This is most useful when you are working on the definition of a new mode.

The *magnification* and *mfname* arguments are explained in Section 6.1 [mf invocation], page 26. In the file `newmode.mf`, you should have the following (with no `mode_def` or `enddef`), if you are using `modes.mf` conventions:

```
mode_param (pixels_per_inch, dpi);
mode_param (blacker, b);
mode_param (fillin, f);
mode_param (o_correction, o);
mode_common_setup_;
```

(Of course, you should use real numbers for *dpi*, *b*, *f*, and *o*.)

For more information on the use of `smode`, or if you are not using `modes.mf`, see page 269 of *The Metafontbook*.

6.4 Online Metafont graphics

The Web2c implementation of Metafont can do online graphics with a number of devices. (See the Metafont manual for more information about how to draw on your screen.) By default, no graphics support is enabled.

Metafont examines the `MFTERM` environment variable or config file value at runtime, or the `TERM` environment variable if `MFTERM` is not set, to determine the device support to use. Naturally, only the devices for which support has been compiled in can be selected.

Here is a table of the possibilities, showing the `MFTERM` value and the corresponding `configure` option(s) in parentheses.

`epsf`
: (‘`--enable-epsfwin`’) Pseudo-window server for Encapsulated PostScript (see `web2c/window/epsf.c`). This device produces an EPS file containing the graphics which would be displayed online on other devices. The name of the EPS file defaults to metafont.eps but can be changed by setting the MFEPSF environment variable to the new filename. Contributed by Mathias Herberts.

`hp2627`
: (‘`--enable-hp2627win`’) HP2627a color graphics terminals.

`mftalk`
: (‘`--enable-mftalkwin`’) Generic window server (see `web2c/window/mftalk.c`).

`next`
: (‘`--enable-next`’) NeXT window system. This requires a separate program, called `DrawingServant`, available separately. See the `web2c/window/next.c`.

`regis`
: (‘`--enable-regiswin`’) Regis terminals.

`sun`
: (‘`--enable-suntoolswin`’) The old Suntools (not any flavor of X) window system. (You can get the even older SunWindows `gfx` system by using `sun-gfx.c`.)

`tek`
: (‘`--enable-tektronixwin`’) Tektronix terminals.

`uniterm`
: (‘`--enable-unitermwin`’) Uniterm, Simon Poole’s emulator of a smart Tektronix 4014 terminal. This may work with regular Tektronix terminals as well; it’s faster than the driver ‘`--enable-tektronixwin`’ selects.

`xterm`
: ‘`--with-x`’ The X window system (version 11).

  There are two variants of the X11 support, one that works with the Xt toolkit, and another that works directly with Xlib. The Xt support is more efficient and has more functionality, so it is the default. If you must use the Xlib support, use ‘`configure --with-x --with-kf-x-toolkit=no`’.

  Specify ‘`--disable-mf-nowin`’ in order not to build a separate non-windows-capable Metafont executable `mf-nowin` (or `mf-nowin.exe`).

  You cannot specify any of the usual X options (e.g., ‘`-geometry`’) on the Metafont command line, but you can specify X resources in your `~/.Xdefaults` or `~/.Xresources` file. The class name is `Metafont`. If you’re using the Xt support, all the usual X toolkit resources are supported. If you’re using the Xlib support, only the `geometry` resource is supported.

  You specify the X display to which Metafont connects in the `DISPLAY` environment variable, as usual.

Writing support for a new device is straightforward. Aside from defining the basic drawing routines that Metafont uses (see `mf.web`), you only have to add another entry to the tables on the last page of `web2c/lib/texmfmp.c`. Or you can write an independent program and use MFtalk (see `web2c/window/mftalk.c`).

6.5 GFtoDVI: Character proofs of fonts

GFtoDVI makes *proof sheets* from a GF bitmap file as output by, for example, Metafont (see Chapter 6 [Metafont], page 26). This is an indispensable aid for font designers or Metafont hackers. Synopsis:

```
gftodvi [option]... gfname[gf]
```

The font *gfname* is searched for in the usual places (see Section "Glyph lookup" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

The suffix '`gf`' is supplied if not already present. This suffix is not an extension, no '`.`' precedes it; for instance, `cmr10.600gf`.

The output filename is the basename of *gfname* extended with `.dvi`, e.g., '`gftodvi /wherever/foo.600gf`' creates `./foo.dvi`.

The characters from *gfname* appear one per page in the DVI output, with labels, titles, and annotations, as specified in Appendix H (Hardcopy Proofs) of *The Metafontbook*.

GFtoDVI uses several fonts besides *gfname* itself:

- *gray font* (default `gray`): for the pixels that actually make up the character. Simply using black is not right, since then labels, key points, and other information could not be shown.
- *title font* (default `cmr8`): for the header information at the top of each output page.
- *label font* (default `cmtt10`): for the labels on key points of the figure.
- *slant font* (no default): for diagonal lines, which are otherwise simulated using horizontal and vertical rules.

To change the default fonts, you must use `special` commands in your Metafont source file, typically via commands like `slantfont slantlj4`. There is no default slant font since no one printer is suitable as a default. You can make your own by copying one of the existing files, such as `.../fonts/source/public/misc/slantlj4.mf` and then running `mf` on it.

For testing purposes, you may it useful to run `mf-nowin rtest` (hit RETURN when it stops) to get a `gf` file of a thorn glyph. Or use `mf` instead of `mf-nowin` to have the glyph(s) displayed on the screen. After that, `gftodvi rtest.2602gf` should produce `rtest.dvi`, which you process as usual.

The program accepts the following option, as well as the standard '`-verbose`', '`-help`', and '`-version`' (see Section 3.2 [Common options], page 6):

'`-overflow-label-offset=`*`points`*'
: Typeset the so-called overflow labels, if any, *points* TEX points from the right edge of the character bounding box. The default is a little over two inches (ten million scaled points, to be precise). Overflow equations are used to locate coordinates when their actual position is too crowded with other information.

6.6 MFT: Prettyprinting Metafont source

MFT translates a Metafont program into a TeX document suitable for typesetting, with the aid of TeX macros defined in the file `mftmac.tex`. Synopsis:

```
mft [option]... mfname[.mf]
```

MFT searches the usual places for *mfname* (see Section "Supported file formats" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program. The output goes to the basename of *mfname* extended with `.tex`, e.g., '`mft /wherever/foo.mf`' creates `./foo.tex`.

Line breaks in the input are carried over into the output; moreover, blank spaces at the beginning of a line are converted to quads of indentation in the output. Thus, you have full control over the indentation and line breaks. Each line of input is translated independently of the others.

Further control is allowed via Metafont comments:

- Metafont comments following a single '%' should be valid TeX input. But Metafont material can be included within vertical bars in a comment; this will be translated by MFT as if it were regular Metafont code. For example, a comment like '`% |x2r| is the tip of the bowl`' will be translated into the TeX '`% $x_{2r}$ is the ...`', i.e., the '`x2r`' is treated as an identifier.
- '%%' indicates that the remainder of an input line should be copied verbatim to the output. This is typically used to introduce additional TeX material at the beginning or an MFT job, e.g. code to modify the standard layout or the formatting macros defined in `mftmac.tex`, or to add a line saying '`%%\bye`' at the end of the job. (MFT doesn't add this automatically in order to allow processing several files produces by MFT in the same TeX job.)
- '`%%% token1 other-tokens`' introduces a change in MFT's formatting rules; all the *other-tokens* will henceforth be translated according to the current conventions for *token1*. The tokens must be symbolic (i.e., not numeric or string tokens). For example, the input line

  ```
  %%% addto fill draw filldraw
  ```

  says to format the '`fill`', '`draw`', and '`filldraw`' operations of plain Metafont just like the primitive token '`addto`', i.e., in boldface type. Without such reformatting commands, MFT would treat '`fill`' like an ordinary tag or variable name. In fact, you need a '%%%' command even to get parentheses to act like delimiters.
- '%%%%' introduces an MFT comment, i.e., MFT ignores the remainder of such a line.
- Five or more '%' signs should not be used.

(The above description was edited from `mft.web`, written by D.E. Knuth.)

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-change=chfile[.ch]`'
: Apply the change file *chfile* as with Tangle and Weave (see Chapter 9 [WEB], page 38).

'`-style=`*`mftfile`*`[.mft]`'

Read *mftfile* before anything else; a MFT style file typically contains only MFT directives as described above. The default style file is named `plain.mft`, which defines this properly for programs using plain Metafont. The MFT files is searched along the `MFTINPUTS` path; see Section "Supported file formats" in *Kpathsea*.

Other examples of MFT style files are `cmbase.mft`, which defines formatting rules for the macros defined in `cm.base`, and `e.mft`, which was used in the production of Knuth's Volume E, *Computer Modern Typefaces*.

Using an appropriate MFT style file, it is also possible to configure MFT for typesetting MetaPost sources. However, MFT does not search the usual places for MetaPost input files.

If you use eight-bit characters in the input file, they are passed on verbatim to the TEX output file; it is up to you to configure TEX to print these properly.

7 MetaPost: Generating PostScript

MetaPost is a picture-drawing language similar to Metafont (see Chapter 6 [Metafont], page 26), but instead of outputting bitmaps in a "font", it outputs PostScript commands. It's primarily intended for creating technical illustrations, but can also be used to create PostScript or OpenType fonts (`https://ctan.org/pkg/metatype1`).

MetaPost also provides for arbitrary integration of text and graphics in a natural way, using any typesetter (TEX and Troff are both supported) and a number of other subsidiary programs, described below.

7.1 `mpost` invocation

MetaPost (installed as `mpost`) reads a series of pictures specified in the MetaPost programming language, and outputs corresponding PostScript code. This section merely describes the options available in the Web2c implementation. For a complete description of the MetaPost language, see AT&T technical report CSTR-162, generally available in *texmf*`/doc/metapost/`, where *texmf* is the root of TEX directory structure. The MetaPost home page: `https://tug.org/metapost`.

Also, a standard MetaPost package for drawing graphs is documented in AT&T technical report CSTR-164, available as the file `mpgraph.ps`, generally stored alongside `mpman.ps`.

MetaPost processes its command line and determines its memory dump (mem) file in a way analogous to Metafont and TEX (see Section 5.1 [`tex` invocation], page 16, and see Section 4.3 [Memory dumps], page 11). Synopses:

```
mpost [option]... [mpname[.mp]] [mp-commands]
mpost [option]... \first-line
mpost [option]... &mem args
```

MetaPost searches the usual places for the main input file *mpname* (see Section "Supported file formats" in *Kpathsea*), extending *mpname* with `.mp` if necessary. To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

MetaPost writes its PostScript output to a series of files *`basempname.nnn`* (or perhaps *`basempname`*`.ps`, very occasionally *`basempname`*`.tfm`), where *nnn* are the figure numbers specified in the input, typically to the `beginfig` macro, and *basempname* is the basename of *mpname*, or '`mpout`' if no input file was specified. MetaPost uses the '`.ps`' extension when the figure number is out of range, e.g., if you say `beginfig(-1)`.

You can use the output files as figures in a TEX document just as with any other PostScript figures. For example, with this TEX command:

```
\special{psfile="filename"}
```

or by using `epsf.tex` (see Section "EPSF macros" in *Dvips*).

The MetaPost construct

```
btex ... tex-input ... etex
```

generates a MetaPost picture expression corresponding to *tex-input*.

The construct

```
verbatimtex ... tex-input ... etex
```

simply passes the *tex-input* through to TEX. For example, if you are using LATEX, your MetaPost input file must start with a `verbatimtex` block that gives the necessary `\documentclass` (or `\documentstyle`) `\begin{document}` command. You will also need to set the enviroment variable `TEX` to '`latex`'.

tex-input need not be specifically TEX input; it could also be Troff. In that case, you will need the '`-m pictures`' Troff macro package (unfortunately absent from many Troff implementations), or an equivalent such as the '`-m pspic`' macros from GNU groff described in grops(1).

Naturally, you must use fonts that are supported by the typesetter; specifically, you'll probably want to use standard PostScript fonts with Troff. And only the TEX system understands Computer Modern or other Metafont fonts; you can also use PostScript fonts with TEX, of course.

MetaPost-generated PostScript figures which do use Computer Modern fonts for labels cannot be directly previewed or printed. Instead, you must include them in a TEX document and run the resulting DVI file through Dvips to arrange for the downloading of the required fonts (see Section "Fonts in figures" in *Dvips*). To help with this, the MetaPost distribution provides a small TEX file `mproof.tex` which is typically called as:

```
    tex mproof mp-output-files... ; dvips mproof -o
```

The resulting file `mproof.ps` can then be printed or previewed.

To generate EPSF files, set the internal MetaPost variable `prologues` positive. To make the output files self-contained, use only standard PostScript fonts. MetaPost reads the same `psfonts.map` file as Dvips, to determine PostScript fonts that need to be downloaded (see Section "psfonts.map" in *Dvips*).

It is posible for pdfTEX to read MetaPost output directly; this is in contrast to general EPSF files, which have to be converted for use with PDF output. The easiest way is to name the MetaPost output files with the `.mps` extension. Then the LATEX `\includegraphics` command, for example, will be able to read them, even when outputting PDF.

MetaPost can write output files, via the `write` primitive; this opens a security hole. See Section 5.1 [tex invocation], page 16.

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-[no]-file-line-error`'
'`-fmt=`*fmtname*'
'`-halt-on-error`'
'`-ini`'
'`-interaction=`*string*'
'`-jobname=`*string*'
'`-kpathsea-debug=`*number*'
'`-[no]parse-first-line`'
'`-output-directory`'
'`-progname=`*string*'
'`-recorder`'
'`-translate-file=`*tcxfile*'
'`-8bit`' These options are common to TEX, Metafont, and MetaPost. See Section 3.2 [Common options], page 6.

‘-T’
‘-troff’ Set the prologues internal variable to 1.

‘-tex=*texprogram*’
When this option is given, the program *texprogram* is used to typeset the labels.

7.2 Initial MetaPost

As of MetaPost 1.504 (TEX Live 2011), MetaPost no longer dumps .mem files (see Section 4.3 [Memory dumps], page 11) and does not distinguish virgin and initial forms (see Section 4.2 [Initial and virgin], page 11). Instead, the “initial” file name is read in its source form—that is, mpost.mp when the program is invoked as mpost.

For a list of options and other information, see Section 7.1 [mpost invocation], page 33.

MetaPost provides a format with all the features of plain Metafont, called mfplain. You can use that in the same way; just run mfplain instead of mpost. This lets you directly process Metafont source files with MetaPost, producing character proofs (one file for each character) similar to those produced with Metafont in proof mode and GFtoDVI (see Section 6.5 [gftodvi invocation], page 30).

7.3 DVItoMP: DVI to MPX conversion

DVItoMP converts DVI files into low-level MetaPost commands in a so-called MPX file. Synopsis:

```
dvitomp dvifile[.dvi] [mpxfile[.mpx]]
```

If *mpxfile* is not specified, the output goes to the basename of *dvifile* extended with .mpx, e.g., ‘dvitomp /wherever/foo.dvi’ creates ./foo.mpx.

DVItoMP supports Dvips-style color specials, such as ‘color push *name*’ and ‘color pop’, outputting them as withcolor MetaPost commands.

The only options are ‘-help’ and ‘-version’ (see Section 3.2 [Common options], page 6).

8 BibTEX: Bibliographies

BibTEX automates much of the job of typesetting bibliographies, and makes bibliography entries reusable in many different contexts.

8.1 BibTEX invocation

BibTEX creates a printable bibliography (`.bbl`) file from references in a `.aux` file, generally written by TEX or LATEX. The `.bbl` file is then incorporated on a subsequent run. The basic bibliographic information comes from `.bib` files, and a BibTEX style (`.bst`) file controls the precise contents of the `.bbl` file. Synopsis:

```
bibtex [option]... auxfile[.aux]
```

The output goes to the basename of *auxfile* extended with `.bbl`; for example, '`bibtex /wherever/foo.aux`' creates `./foo.bbl`. BibTEX also writes a log file to the basename of *auxfile* extended with '`.blg`'.

The names of the `.bib` and `.bst` files are specified in the `.aux` file as well, via the `\bibliography` and `\bibliographystyle` (La)TEX macros. BibTEX searches for `.bib` files using the `BIBINPUTS` and `TEXBIB` paths, and for `.bst` files using `BSTINPUTS` (see Section "Supported file formats" in *Kpathsea*). It does no path searching for `.aux` files.

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-terse`' Suppress the program banner and progress reports normally output.

'`-min-crossrefs=`*n*'

If at least *n* (2 by default) bibliography entries refer to another entry *e* via their `crossref` field, include *e* in the `.bbl` file, even if it was not explicitly referenced in the `.aux` file. For example, *e* might be a conference proceedings as a whole, with the cross-referencing entries being individual articles published in the proceedings.

If you want to avoid these automatic inclusions altogether, make *n* a sufficiently large number, and be sure to remove any previous `.aux` and `.bbl` files. Otherwise the option may appear to have no effect, since BibTEX will have added the citation for *e* to the `.aux`, and nothing will remove it.

See also:

`btxdoc.tex`

Basic LATEXable documentation for general BibTEX users.

`btxhak.tex`

LATEXable documentation for style designers.

`btxdoc.bib`

BibTEX database file for the two above documents.

`xampl.bib`

Example database file with all the standard entry types.

`ftp://ftp.math.utah.edu/pub/tex/bib/`

A very large `.bib` and `.bst` collection, including references for all the standard TEX books and a complete bibliography for TUGboat.

8.2 Basic BibTEX style files

Here are descriptions of the four standard and four semi-standard basic BibTEX styles. *CTAN:*`/biblio/bibtex` contains these and many more (for CTAN info, see Section "unix-tex.ftp" in *Kpathsea*).

`plain` Sorts entries alphabetically, with numeric labels. Generally formatted according to van Leunen's *A Handbook for Scholars*. The other style files listed here are based on `plain`.

`abbrv` First names, month names, and journal names are abbreviated.

`acm` Names are printed in small caps.

`alpha` Alphanumeric labels, e.g., '`Knu66`'.

`apalike` No labels at all; instead, the year appears in parentheses after the author. Use this in conjunction with `apalike.tex` (plain TEX) or `apalike.sty` (LATEX), which also changes the citations in the text to be '`(`*`author`*`, `*`year`*`)`'.

`ieeetr` Numeric labels, entries in citation order, IEEE abbreviations, article titles in quotes.

`siam` Numeric labels, alphabetic order, *Math. Reviews* abbreviations, names in small caps.

`unsrt` Lists entries in citation order, i.e., unsorted.

`btxbst.doc`
The template file and documentation for the standard styles.

9 WEB: Literate programming

WEB languages allow you to write a single source file that can produce both a compilable program and a well-formatted document describing the program in as much detail as you wish to prepare. Writing in this kind of dual-purpose language is called *literate programming*. (The Usenet newsgroup `comp.programming.literate` is devoted to this subject.)

WEB-like languages have been implemented with many pairs of base languages: Cweb provides C and Troff (see Appendix B [References], page 55); CWEB provides C and TEX (*CTAN:*`/web/c_cpp/cweb`); Spiderweb provides C, C++, Awk, Ada, many others, and TEX (*CTAN:*`/web/spiderweb`); and, of course, the original WEB provides Pascal and TEX, the implementation languages for the original TEX, Metafont, MetaPost, and related programs to come from the TEX project at Stanford.

The original WEB language is documented in the file `webman.tex`, which is included in the `ftp://ftp.tug.org/tex/lib.tar.gz` archive (and available in many other places, of course).

9.1 Tangle: Translate WEB to Pascal

Tangle creates a compilable Pascal program from a WEB source file (see Chapter 9 [WEB], page 38). Synopsis:

```
tangle [option]... webfile[.web] [changefile[.ch]]
```

The Pascal output is written to the basename of *webfile* extended with '`.p`'; for example, '`tangle /wherever/foo.web`' creates `./foo.p`. Tangle applies *changefile* to *webfile* before writing the output; by default, there is no change file.

If the program makes use of the WEB string facility, Tangle writes the string pool to the basename of *webfile* extended with '`.pool`'.

The Pascal output is packed into lines of 72 characters or less, with the only concession to readability being the termination of lines at semicolons when this can be done conveniently.

The program accepts the following options, as well as the standard '`--help`' and '`--version`' (see Section 3.2 [Common options], page 6):

'`-length=`*number*'
: The number of characters that are considered significant in an identifier. Whether underline characters are counted depends on the '`-underline`' option. The default value is 32, the original tangle used 7, but this proved too restrictive for use by Web2c.

'`-lowercase`'
'`-mixedcase`'
'`-uppercase`'
: These options specify the case of identifiers in the output of tangle. If '`-uppercase`' ('`-lowercase`') is specified, tangle will convert all identfiers to uppercase (lowercase). The default is '`-mixedcase`', which specifies that the case will not be changed.

'`-underline`'
: When this option is given, tangle does not strip underline characters from identifiers.

'`-loose`'

'`-strict`' These options specify how strict tangle must be when checking identifiers for equality. The default is '`-loose`', which means that tangle will follow the rules set by the case-smashing and underline options above. If '`-strict`' is set, then identifiers will always be stripped of underlines and converted to uppercase before checking whether they collide.

9.2 Weave: Translate WEB to TEX

Weave creates a TEX document from a WEB source file (see Chapter 9 [WEB], page 38), assuming various macros defined in `webmac.tex`. It takes care of typographic details such as page layout, indentation, and italicizing identifiers. It also automatically gathers and outputs extensive cross-reference information. Synopsis:

```
weave [option]... webfile[.web] [changefile[.ch]]
```

The output is to the basename of *webfile* extended with '`.tex`'; for example, '`weave /wherever/foo.web`' creates `./foo.tex`. Weave applies *changefile* to *webfile* before writing the output; by default, there is no change file.

The program accepts the following option, as well as the standard '`-verbose`', '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-x`' Omit the cross-reference information: the index, the list of WEB module names, and the table of contents (an empty `CONTENTS.tex` file will still be written when the Weave output file is processed by TEX using the default `webmac.tex`, though).

Conventionally, WEB programmers should define the TEX `\title` macro at the beginning of the source file. Also, to get output of only changed modules, one can say `\let\maybe=\iffalse` (usually as the first change in the change file).

9.3 Pooltype: Display WEB pool files

Pooltype shows the so-called *string number* of each string in a WEB pool file (see Chapter 9 [WEB], page 38), as output by Tangle (see Section 9.1 [tangle invocation], page 38), including the first 256 strings corresponding to the possible input characters. Pooltype primarily serves as an example of WEB conventions to implementors of the TEX system. Synopsis:

```
pooltype [option]... poolfile[.pool]
```

No path searching is done for *poolfile*. Output is to standard output.

The only options are '`--help`' and '`--version`' (see Section 3.2 [Common options], page 6).

As an example of the output, here is the (edited) output for `tex.pool`:

```
0: "^^@"
1: "^^A"
...
255: "^^ff"
256: "pool size"
...
1314: "Using character substitution: "
```

```
    (23617 characters in all.)
```

In Metafont and MetaPost, the first 256 characters are actually represented as single bytes (i.e., themselves), not in the '^^' notation. Consider Pooltype as showing the results after conversion for output.

10 DVI utilities

TEX outputs a file in *DVI* (DeVice Independent) format as a compact representation of the original document. DVI files can be translated to meet the requirements of a real physical device, such as PostScript printers (see Section “Introduction” in *Dvips*), PCL printers (see dvilj(1)), and X displays (see xdvi(1)). In fact, DVI translators are available for virtually all common devices: see *CTAN:*`/dviware` (for CTAN info, see Section “unixtex.ftp” in *Kpathsea*).

For the precise definition of the DVI file format, see (for example) the source file `web2c/dvitype.web`.

The DVI-processing programs in the Web2c distribution are not device drivers; they perform generic utility functions.

10.1 DVIcopy: Canonicalize virtual font references

DVIcopy reads a DVI file, expands any references to virtual fonts (see Section “Virtual fonts” in *Dvips*) to base fonts, and writes the resulting DVI file. Thus you can use virtual fonts even if your DVI processor does not support them, by passing the documents through DVIcopy first. Synopsis:

```
dvicopy [option]... [indvi[.dvi] [outdvi[.dvi]]]
```

DVIcopy reads standard input if *indvi* is not specified, and writes standard output if *outdvi* is not specified.

The program accepts the following options, as well as the standard ‘`-help`’ and ‘`-version`’ (see Section 3.2 [Common options], page 6):

‘`-magnification=`*integer*’
: Override existing magnification in *indvi* with *integer*; 1000 specifies no magnification. This is equivalent to setting TEX’s `\mag` parameter.

‘`-max-pages=`*n*’
: Process *n* pages; default is one million.

‘`-page-start=`*page-spec*’
: Start at the first page matching *page-spec*, which is one or more (signed) integers separated by periods, corresponding to TEX’s `\count0...9` parameters at `\shipout` time; ‘`*`’ matches anything. Examples: ‘`3`’, ‘`1.*.-4`’.

10.2 DVItype: Plain text transliteration of DVI files

DVItype translates a DeVice Independent (DVI) file (as output by TEX, for example) to a plain text file that humans can read. It also serves as a DVI-validating program, i.e., if DVItype can read a file, it’s correct. Synopsis:

```
dvitype [option]... dvifile[.dvi]
```

DVItype does not read any bitmap files, but it does read TFM files for fonts referenced in *dvifile*. The usual places are searched (see Section “Supported file formats” in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to ‘`-1`’ before running the program.

Output goes to standard output.

The program accepts the following options, as well as the standard '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-dpi=`*`real`*'
: Do pixel movement calculations at *real* pixels per inch; default 300.0.

'`-magnification=`*`integer`*'
: Override existing magnification in *indvi* with *integer*; 1000 specifies no magnification. This is equivalent to setting TEX's `\mag` parameter.

'`-max-pages=`*`n`*'
: Process *n* pages; default is one million.

'`-output-level=`*`n`*'
: Verbosity level of output, from 0 to 4 (default 4):

  - 0: Global document information only.
  - 1: Most DVI commands included, and typeset characters summarized.
  - 2: Character and movement commands explicitly included.
  - 3: DVI stack and current position calculations included.
  - 4: Same information as level 3, but DVItype does random positioning in the file, reading the DVI postamble first.

'`-page-start=`*`page-spec`*'
: Start at the first page matching *page-spec*, which is one or more (signed) integers separated by periods, corresponding to TEX's `\count0...9` parameters at `\shipout` time; '`*`' matches anything. Examples: '`1`', '`5.*.-9`'.

'`-show-opcodes`'
: Show numeric opcode values (in decimal) for DVI commands, in braces after the command name. This can help in debugging DVI utilities. We use decimal because in the DVI format documentation (in `dvitype.web`, among others) the opcodes are shown in decimal.

10.2.1 DVItype output example

As an example of the output from DVItype (see section above), here is its (abridged) translation of the `story.dvi` resulting from running the example in *The TEXbook*, with '`-output-level=4`' and '`-show-opcodes`' on.

```
...
Options selected:
  Starting page = *
  Maximum number of pages = 1000000
  Output level = 4 (the works)
  Resolution = 300.00000000 pixels per inch
numerator/denominator=25400000/473628672
magnification=1000;       0.00006334 pixels per DVI unit
' TeX output 1992.05.17:0844'
Postamble starts at byte 564.
```

```
maxv=43725786, maxh=30785863, maxstackdepth=3, totalpages=1
Font 33: cmsl10---loaded at size 655360 DVI units
Font 23: cmbx10---loaded at size 655360 DVI units
Font 0: cmr10---loaded at size 655360 DVI units

42: beginning of page 1
87: push {141}
level 0:(h=0,v=0,w=0,x=0,y=0,z=0,hh=0,vv=0)
88: down3 -917504 {159} v:=0-917504=-917504, vv:=-58
92: pop {142}
...
104: putrule {137} height 26214, width 30785863 (2x1950 pixels)
113: down3 5185936 {159} v:=655360+5185936=5841296, vv:=370
117: push {141}
level 1:(h=0,v=5841296,w=0,x=0,y=0,z=0,hh=0,vv=370)
118: right4 12265425 {146} h:=0+12265425=12265425, hh:=777
[ ]
123: fntdef1 23 {243}: cmbx10
145: fntnum23 {194} current font is cmbx10
146: setchar65 h:=12265425+569796=12835221, hh:=813
147: w3 251220 {150} h:=12835221+251220=13086441, hh:=829
151: setchar83 h:=13086441+418700=13505141, hh:=856
...
164: setchar82 h:=17448202+565245=18013447, hh:=1142
165: x0 -62805 {152} h:=18013447-62805=17950642, hh:=1138
166: setchar89 h:=17950642+569796=18520438, hh:=1174
[A SHORT STORY]
167: pop {142}
level 1:(h=0,v=5841296,w=0,x=0,y=0,z=0,hh=0,vv=370)
...
550: pop {142}
level 0:(h=0,v=42152922,w=0,x=0,y=0,z=0,hh=0,vv=2670)
551: down3 1572864 {159} v:=42152922+1572864=43725786, vv:=2770
555: push {141}
level 0:(h=0,v=43725786,w=0,x=0,y=0,z=0,hh=0,vv=2770)
556: right4 15229091 {146} h:=0+15229091=15229091, hh:=965
561: setchar49 h:=15229091+327681=15556772, hh:=986
[ 1]
562: pop {142}
level 0:(h=0,v=43725786,w=0,x=0,y=0,z=0,hh=0,vv=2770)
563: eop {140}
```

Explanation:

- The DVItype options are recorded at the beginning, followed by global information about the document, including fonts used.
- Each DVI command is preceded by its byte position in the file (‘`42:`’, ‘`87:`’, . . .),

and (because of the '`-show-opcodes`') followed by its decimal opcode value in braces ('`{141}`', '`{142}`', ...).

- The '`level`' lines record information about the DVI stack; '`h`' and '`v`' define the current position in DVI units, while '`hh`' and '`vv`' are the same in pixels.
- Text sequences are summarized in brackets, as in '`[A SHORT STORY]`' and the '`[ 1]`'.

11 Font utilities

The Web2c programs described here convert between various TeX-related font formats; the first section below briefly describes the formats. GFtoPK is the only one that is routinely used, as Metafont outputs GF format, but it's most efficient for device drivers to use PK.

The precise definitions of the PK, GF, TFM, PL, VF, and VPL formats mentioned below are in the source files that read them; `pktype.web`, `gftype.web`, `tftopl.web`, etc.

11.1 Font file formats

(For another perspective on this, see Section "Font concepts" in *Dvips*).

Font files come in several varieties, with suffixes like:

`.tfm` `.*pk` `.*gf` `.*pxl` (obsolete) `.pl` `.mf` `.vf` `.vpl`

Each represents a file format.

A TFM (TeX font metric) file is a compact binary file that contains information about each character in a font, about combinations of characters within that font, and about the font as a whole. The font metric information contained in TFM files is device-independent units is used by TeX to do typesetting. Unlike the bitmap (raster) fonts described below, TFM font files contain no information about the shapes of characters. They describe rectangular areas and combinations thereof, but not what will eventually be printed in those areas.

Since TeX does scaling calculations, one TFM file serves for all magnifications of a given typeface. On the other hand, the best printed results are obtained when magnified (or reduced fonts) are not produced geometrically (as done by PostScript, for example) but rather optically, with each size a separate design (as done with Computer Modern and the EC fonts, for example); then a separate TFM file is needed for each size.

At any rate, TeX produces a DVI (DeVice Independent) file from your source document. In order to print DVI files on real devices, you need font files defining digitized character shapes and other data. Then previewers and printer-driver programs can translate your DVI files into something usable by your monitor or printer. Bitmap fonts come with suffixes such as '`.600pk`' or '`.600gf`' or '`.3000pxl`', where the '`600`' is the horizontal dots-per-inch resolution at which the font was produced, and the '`pk`' or '`gf`' or '`pxl`' indicates the font format. Outline fonts in PostScript Type 1 format have suffixes such as '`.pfa`' or '`.pfb`'.

Fonts in pk (packed) format are in the tightly packed raster format that is pretty much the standard today. They take up less space than fonts in the gf (generic font) format that Metafont generates, and far less space than fonts in pxl format. Fonts in pxl format take up gross amounts of disk space and permit only 128 characters. They are obsolete.

Font files with the '`.pl`' (property list) suffix are the plain text (human-readable) analog of the binary '`.tfm`' files. The TFtoPL and PLtoTF programs convert between the two formats (see Section 11.6 [tftopl invocation], page 50, and Section 11.7 [pltotf invocation], page 52).

Font files with the '`.mf`' suffix are in Metafont source format. These are the files used by Metafont to generate rastered fonts for specific typefaces at specific magnifications for the specific resolution and type of mapping used by your device.

The suffix '`.vf`' identifies "virtual font" files, for which '`.vpl`' is the human-readable analog. See See Section 11.8 [vftovp invocation], page 52, and Section 11.9 [vptovf invocation], page 53. For further discussion of virtual fonts, see *CTAN:*`/doc/virtual-fonts.knuth`, *CTAN:*`/help/virtualfonts.txt`, and Section "Virtual fonts" in *Dvips*.

(This section is based on documentation in the original Unix TEX distribution by Pierre MacKay and Elizabeth Tachikawa.)

11.2 GFtoPK: Generic to packed font conversion

GFtoPK converts a generic font (GF) file output by, for example, Metafont (see Section 6.1 [mf invocation], page 26) to a packed font (PK) file. PK files are considerably smaller than the corresponding gf files, so they are generally the bitmap font format of choice. Some DVI-processing programs, notably Dvips, only support PK files and not GF files. Synopsis:

```
gftopk [option]... gfname.dpi[gf] [pkfile]
```

The font *gfname* is searched for in the usual places (see Section "Glyph lookup" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

The suffix '`gf`' is supplied if not already present. This suffix is not an extension; no '`.`' precedes it: for instance, `cmr10.600gf`.

If *pkfile* is not specified, the output is written to the basename of '*`gfname.dpi`*`pk`', e.g., '`gftopk /wherever/cmr10.600gf`' creates `./cmr10.600pk`.

The only options are '`--verbose`', '`--help`', and '`--version`' (see Section 3.2 [Common options], page 6).

11.3 PKtoGF: Packed to generic font conversion

PKtoGF converts a packed font (PK) file to a generic font (GF) file. Since PK format is much more compact than GF format, the most likely reason to do this is to run GFtype (see Section 11.5 [gftype invocation], page 48) on the result, so you can see the bitmap images. Also, a few old utility programs do not support PK format. Synopsis:

```
pktogf [option]... pkname.dpi[pk] [gffile]
```

The font *pkname* is searched for in the usual places (see Section "Glyph lookup" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

The suffix '`pk`' is supplied if not already present. This suffix is not an extension; no '`.`' precedes it: for instance, `cmr10.600pk`.

If *gffile* is not specified, the output is written to the basename of '*`pkname.dpi`*`gf`', e.g., '`pktogf /wherever/cmr10.600pk`' creates `./cmr10.600gf`.

The only options are '`--verbose`', '`--help`', and '`--version`' (see Section 3.2 [Common options], page 6).

11.4 PKtype: Plain text transliteration of packed fonts

PKtype translates a packed font (PK) bitmap file (as output by GFtoPK, for example) to a plain text file that humans can read. It also serves as a PK-validating program, i.e., if PKtype can read a file, it's correct. Synopsis:

```
pktype pkname.dpi[pk]
```

The font *pkname* is searched for in the usual places (see Section "Glyph lookup" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

The suffix '`pk`' is supplied if not already present. This suffix is not an extension; no '`.`' precedes it: for instance, `cmr10.600pk`.

The translation is written to standard output.

The only options are '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6).

As an example of the output, here is the (abridged) translation of the letter 'K' in '`cmr10`', as rendered at 600 dpi with the mode '`ljfour`' from `modes.mf` (available from `ftp://ftp.tug.org/tex/modes.mf`).

```
955:  Flag byte = 184  Character = 75  Packet length = 174
  Dynamic packing variable = 11
  TFM width = 815562  dx = 4259840
  Height = 57  Width = 57  X-offset = -3  Y-offset = 56
  [2]23(16)17(8)9(25)11(13)7(27)7(16)7(28)4(18)7(28)2(20)7(27)...
  ...
  (14)9(24)12(5)[2]23(13)21
```

Explanation:

'`955`' The byte position in the file where this character starts.

'`Flag byte`'
'`Dynamic packing variable`'
: Related to the packing for this character; see the source code.

'`Character`'
: The character code, in decimal.

'`Packet length`'
: The total length of this character definition, in bytes.

'`TFM width`'
: The device-independent (TFM) width of this character. It is 2^24 times the ratio of the true width to the font's design size.

'`dx`' The device-dependent width, in *scaled pixels*, i.e., units of horizontal pixels times 2^16.

'`Height`'
'`Width`' The bitmap height and width, in pixels.

'`X-offset`'
'`Y-offset`'
: Horizontal and vertical offset from the upper left pixel to the reference (origin) pixel for this character, in pixels (right and down are positive). The *reference pixel* is the pixel that occupies the unit square in Metafont; the Metafont reference point is the lower left hand corner of this pixel. Put another way, the x-offset is the negative of the left side bearing; the right side bearing is the horizontal escapement minus the bitmap width plus the x-offset.

‘`[2]23(16)...`’
: Finally, run lengths of black pixels alternate with parenthesized run lengths of white pixels, and brackets indicate a repeated row.

11.5 GFtype: Plain text transliteration of generic fonts

GFtype translates a generic font (GF) bitmap file (as output by Metafont, for example) to a plain text file that humans can read. It also serves as a GF-validating program, i.e., if GFtype can read a file, it's correct. Synopsis:

```
gftype [option]... gfname.dpi[gf]
```

The font *gfname* is searched for in the usual places (see Section “Glyph lookup” in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to ‘`-1`’ before running the program.

The suffix ‘`gf`’ is supplied if not already present. This suffix is not an extension; no ‘`.`’ precedes it: for instance, `cmr10.600gf`.

The translation is written to standard output.

The program accepts the following options, as well as the standard ‘`-help`’ and ‘`-version`’ (see Section 3.2 [Common options], page 6):

‘`-images`’ Show the characters’ bitmaps using asterisks and spaces.

‘`-mnemonics`’
: Translate all commands in the GF file.

As an example of the output, here is the (abrdiged) translation of the letter ‘K’ in ‘`cmr10`’, as rendered at 600 dpi with the mode ‘`ljfour`’ from `modes.mf` (available from `ftp://ftp.tug.org/tex/modes.mf`), with both ‘`-mnemonics`’ and ‘`-images`’ enabled.

GFtype outputs the information about a character in two places: a main definition and a one-line summary at the end. We show both. Here is the main definition:

```
2033: beginning of char 75: 3<=m<=60 0<=n<=56
(initially n=56) paint (0)24(12)20
2043: newrow 0 (n=55) paint 24(12)20
2047: newrow 0 (n=54) paint 24(12)20
2051: newrow 0 (n=53) paint 24(12)20
2055: newrow 7 (n=52) paint 10(21)13
2059: newrow 8 (n=51) paint 8(23)9
...
2249: newrow 8 (n=5) paint 8(23)11
2253: newrow 7 (n=4) paint 10(22)12
2257: newrow 0 (n=3) paint 24(11)22
2261: newrow 0 (n=2) paint 24(11)22
2265: newrow 0 (n=1) paint 24(11)22
2269: newrow 0 (n=0) paint 24(11)22
2273: eoc
```

```
.<--This pixel's lower left corner is at (3,57) in METAFONT coordinates
************************            ********************
************************            ********************
************************            ********************
************************            ********************
       **********                     *************
        ********                       *********
...
        ********                       ***********
       **********                      ************
************************           **********************
************************           **********************
************************           **********************
************************           **********************
.<--This pixel's upper left corner is at (3,0) in METAFONT coordinates
```

Explanation:

‘2033’
‘2043’
‘...’ The byte position in the file where each GF command starts.

‘`beginning of char 75`’
The character code, in decimal.

‘`3<=m<=60 0<=n<=56`’
The character’s bitmap lies between 3 and 60 (inclusive) horizontally, and between 0 and 56 (inclusive) vertically. (*m* is a column position and *n* is a row position.) Thus, 3 is the left side bearing. The right side bearing is the horizontal escapement (given below) minus the maximum *m*.

‘`(initially n=56) paint (0)24(12)20`’
The first row of pixels: 0 white pixels, 24 black pixels, 12 white pixels, etc.

‘`newrow 0 (n=55) paint 24(12)20`’
The second row of pixels, with zero leading white pixels on the row.

‘`eoc`’ The end of the main character definition.

Here is the GF postamble information that GFtype outputs at the end:

```
Character 75: dx 4259840 (65), width 815562 (64.57289), loc 2033
```

Explanation:

‘`dx`’ The device-dependent width, in *scaled pixels*, i.e., units of horizontal pixels times 2^16. The ‘`(65)`’ is simply the same number rounded. If the vertical escapement is nonzero, it would appear here as a ‘`dy`’ value.

‘`width`’ The device-independent (TFM) width of this character. It is 2^24 times the ratio of the true width to the font’s design size. The ‘`64.57289`’ is the same number converted to pixels.

‘`loc`’ The byte position in the file where this character starts.

11.6 TFtoPL: TEX font metric to property list conversion

TFtoPL translates a TEX font metric (TFM, see Section "Metric files" in *Dvips*) file (as output by Metafont, for example) to *property list format* (a list of parenthesized items describing the font) that humans can edit or read. This program is mostly used by people debugging TEX implementations, writing font utilities, etc. Synopsis:

```
tftopl [option]... tfmname[.tfm] [plfile[.pl]]
```

The font *tfmname* (extended with '`.tfm`' if necessary) is searched for in the usual places (see Section "Supported file formats" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program.

If *plfile* (which is extended with '`.pl`' if necessary) is not specified, the property list file is written to standard output. The property list file can be converted back to TFM format by the companion program TFtoPL (see the next section).

The program accepts the following option, as well as the standard '`-verbose`', '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-charcode-format=`*type*'

Output character codes in the PL file according to *type*: either '`octal`' or '`ascii`'. Default is '`ascii`' for letters and digits, octal for all other characters. Exception: if the font's coding scheme starts with '`TeX math sy`' or '`TeX math ex`', all character codes are output in octal.

In '`ascii`' format, character codes that correspond to graphic characters, except for left and right parentheses, are output as a '`C`' followed by the single character: '`C K`', for example. In octal format, character codes are output as the letter '`O`' followed by octal digits, as in '`O 113`' for '`K`'.

'`octal`' format is useful for symbol and other non-alphabetic fonts, where using ASCII characters for the character codes is merely confusing.

As an example of the output, here is the (abridged) property list translation of `cmr10.tfm`:

```
(FAMILY CMR)
(FACE O 352)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 11374260171)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.333334)
   (STRETCH R 0.166667)
   (SHRINK R 0.111112)
   (XHEIGHT R 0.430555)
   (QUAD R 1.000003)
   (EXTRASPACE R 0.111112)
   )
(LIGTABLE
   ...
   (LABEL C f)
   (LIG C i O 14)
   (LIG C f O 13)
   (LIG C l O 15)
   (KRN O 47 R 0.077779)
   (KRN O 77 R 0.077779)
   (KRN O 41 R 0.077779)
   (KRN O 51 R 0.077779)
   (KRN O 135 R 0.077779)
   (STOP)
   ...
   )
...
(CHARACTER C f
   (CHARWD R 0.305557)
   (CHARHT R 0.694445)
   (CHARIC R 0.077779)
   (COMMENT
      (LIG C i O 14)
      (LIG C f O 13)
      (LIG C l O 15)
      (KRN O 47 R 0.077779)
      (KRN O 77 R 0.077779)
      ...
      )
   )
...
```

As you can see, the general format is a list of parenthesized *properties*, nested where necessary.

- The first few items (`FAMILY`, `FACE`, and so on) are the so-called *headerbyte* information from Metafont, giving general information about the font.
- The `FONTDIMEN` property defines the TEX `\fontdimen` values.
- The `LIGTABLE` property defines the ligature and kerning table. `LIG` properties define ligatures: in the example above, an '`f`' (in the '`LABEL`') followed by an '`i`' is a ligature, i.e., a typesetting program like TEX replaces those two consecutive characters by the character at position octal '014 in the current font—presumably the 'fi' ligature. `KRN` properties define kerns: if an '`f`' is followed by character octal '047 (an apostrophe), TEX inserts a small amount of space between them: 0.077779 times the design size the font was loaded at (about three-quarters of a printer's point by default in this case, or .001 inches).
- The `CHARACTER` property defines the dimensions of a character: its width, height, depth, and italic correction, also in design-size units, as explained in the previous item. For our example 'f', the depth is zero, so that property is omitted. TFtoPL also inserts any kerns and ligatures for this character as a comment.

11.7 PLtoTF: Property list to TEX font metric conversion

PLtoTF translates a property list file (as output by TFtoPL, for example) to TEX font metric (TFM, see Section "Metric files" in *Dvips*) format. It's much easier for both programs and humans to create the (plain text) property list files and let PLtoTF take care of creating the binary TFM equivalent than to output TFM files directly. Synopsis:

```
pltotf [option]... plfile[.pl] [tfmfile[.tfm]]
```

If *tfmfile* (extended with '`.tfm`' if necessary) is not specified, the TFM file is written to the basename of '`plfile.tfm`', e.g., '`pltotf /wherever/cmr10.pl`' creates `./cmr10.tfm`. (Since TFM files are binary, writing to standard output by default is undesirable.)

The only options are '`-verbose`', '`-help`', and '`-version`' (see Section 3.2 [Common options], page 6).

For an example of property list format, see the previous section.

11.8 VFtoVP: Virtual font to virtual property lists

VFtoVP translates a virtual font metric (VF, see Section "Virtual fonts" in *Dvips*) file and its accompanying TEX font metric (TFM, see Section "Metric files" in *Dvips*) file (as output by VPtoVF, for example) to *virtual property list format* (a list of parenthesized items describing the virtual font) that humans can edit or read. This program is mostly used by people debugging virtual font utilities. Synopsis:

```
vftovp [option]... vfname[.vf] [tfmname[.tfm] [vplfile[.vpl]]]
```

The fonts *vfname* and *tfmname* (extended with '`.vf`' and '`.tfm`' if necessary) are searched for in the usual places (see Section "Supported file formats" in *Kpathsea*). To see all the relevant paths, set the environment variable `KPATHSEA_DEBUG` to '`-1`' before running the program. If *tfmname* is not specified, *vfname* (without a trailing '`.vf`') is used.

If *vplfile* (extended with '`.vpl`' if necessary) is not specified, the property list file is written to standard output. The property list file can be converted back to VF and TFM format by the companion program VFtoVP (see the next section).

The program accepts the following option, as well as the standard '`-verbose`', '`-help`' and '`-version`' (see Section 3.2 [Common options], page 6):

'`-charcode-format=`*type*'

Output character codes in the PL file according to *type*: either '`octal`' or '`ascii`'. Default is '`ascii`' for letters and digits, octal for all other characters. Exception: if the font's coding scheme starts with '`TeX math sy`' or '`TeX math ex`', all character codes are output in octal.

In '`ascii`' format, character codes that correspond to graphic characters, except for left and right parentheses, are output as a '`C`' followed by the single character: '`C K`', for example. In octal format, character codes are output as the letter '`O`' followed by octal digits, as in '`O 113`' for '`K`'.

'`octal`' format is useful for symbol and other non-alphabetic fonts, where using ASCII characters for the character codes is merely confusing.

11.9 VPtoVF: Virtual property lists to virtual font

VPtoVF translates a virtual property list file (as output by VFtoVP, for example) to virtual font (VF, see Section “Virtual fonts” in *Dvips*) and TeX font metric (TFM, see Section “Metric files” in *Dvips*) files. It’s much easier for both programs and humans to create the (plain text) property list files and let VPtoVF take care of creating the binary VF and TFM equivalents than to output them directly. Synopsis:

```
vptovf [option]... vplfile[.vpl] [vffile[.vf] [tfmfile[.tfm]]]
```

If *vffile* (extended with ‘`.vf`’ if necessary) is not specified, the VF output is written to the basename of ‘`vplfile.vf`’; similarly for *tfmfile*. For example, ‘`vptovf /wherever/ptmr.vpl`’ creates `./ptmr.vf` and `./ptmr.tfm`.

The only options are ‘`-verbose`’, ‘`-help`’, and ‘`-version`’ (see Section 3.2 [Common options], page 6).

11.10 Font utilities available elsewhere

The Web2c complement of font utilities merely implements a few basic conversions. Many other more sophisticated font utilities exist; most are in `CTAN:/fonts/utilities` (for CTAN info, see Section “unixtex.ftp” in *Kpathsea*). Here are some of the most commonly-requested items:

- AFM (Adobe font metric) to TFM conversion: see Section “Invoking afm2tfm” in *Dvips*, and `CTAN:/fonts/utilities/afmtopl`.
- BDF (the X bitmap format) conversion: `ftp://ftp.tug.org/tex/bdf.tar.gz`.
- Creating fonts using MetaPost: MetaType1. `ftp://bop.eps.gda.pl/pub/metatype1`. This is used to create the excellent Latin Modern font family (`CTAN:/fonts/lm`), which extends Computer Modern to a vast repertoire of scripts.
- Editing of bitmap fonts: Xbfe from the GNU font utilities mentioned below; the X BDF-editing programs available from `ftp://ftp.x.org/R5contrib/xfed.tar.Z` and `ftp://ftp.x.org/R5contrib/xfedor.tar.Z`; and finally, if your fonts have only 128 characters, you can use the old `gftopxl`, `pxtoch`, and `chtopx` programs from `ftp://ftp.tug.org/tex/web`.
- Editing of outline fonts: FontForge, `fontforge.sourceforge.net`. This is a very elaborate program with support for many outline formats (Type 1, OpenType, TrueType, . . .), and many advanced font editing features.
- PK bitmaps from PostScript outline fonts: gsftopk from the ‘`xdvi`’ distribution. Alternatively, `ps2pk`, from `CTAN:/fonts/utilities/ps2pk`.
- PostScript Type 1 font format conversion (i.e., between PFA and PFB formats): `https://www.lcdf.org/type`.
- Tracing bitmaps to fitted outlines: Autotrace (`http://autotrace.sourceforge.net`), Potrace (`http://potrace.sourceforge.net`). For Metafont fonts, either of the two programs `mftrace` (`http://www.xs4all.nl/~hanwen/mftrace`) or `textrace` (`http://textrace.sourceforge.net`) make the job easier.
- Virtual font creation: `https://ctan.org/pkg/fontinst`.

Appendix A Legalisms

In general, each file has its own copyright notice stating the copying permissions for that file. Following is a summary.

The Stanford TEX programs and Web2c system itself are in the public domain (`https://tug.org/texlive/copying.html`). The sources may be copied verbatim, or used as the starting point of new software under different names; however, per the wishes of the authors, they should be modified only through a `.ch` file, but this is in the nature of a development request rather than a legal requirement.

MLTEX, pdfTEX, LuaTEX, XeTEX, and all the other derived engines have used various license terms for their additions to the base code, often the GPL (see `https://www.gnu.org/licenses/#GPL` or (for example) the file `web2c/pdftexdir/COPYINGv2`. They also mostly make use of additional libraries with their own (compatible) terms. Please see each program's sources.

The Kpathsea library is covered by the GNU Lesser General Public License (see Section "Introduction" in *Kpathsea*). Therefore, the *binaries* resulting from a standard Web2c compilation are also covered by the LGPL; so if you (re)distribute the binaries, you must also (offer to) distribute the complete source that went into those binaries. See `https://gnu.org/licenses/#LGPL` or the file `kpathsea/COPYING.LESSERv2`.

Appendix B References

1. Kpathsea: See *Kpathsea*.
2. Dvips and Afm2tfm: See *Dvips*.
3. The TEX Users Group: `https://tug.org`. For an introduction to the TEX system, see `https://tug.org/begin.html`.
4. TUGboat, the principal journal for the TEX world: `https://tug.org/TUGboat`.
5. TEX and computer typesetting in general: `ftp://ftp.math.utah.edu/pub/tex/bib/texbook1.bib`.
6. For a bibliography of formal articles and technical reports on the TEX project, see the books *TEX: The Program* or *Metafont: The Program* cited below.
7. [Bil87] Neenie Billawala. Write-white printing engines and tuning fonts with Metafont. *TUGboat*, 8(1):29–32, April 1987. `https://tug.org/TUGboat/tb08-1/tb17billawala.pdf`.
8. [Hob89] John D. Hobby. A Metafont-like system with PS output. *TUGboat*, 10(4):505–512, December 1989. `https://tug.org/metapost`.
9. [Hob92] John D. Hobby. A User's Manual for MetaPost. Technical Report CSTR-162, AT&T Bell Laboratories, 1992.
10. [Hob93] John D. Hobby. Drawing Graphs with MetaPost. Technical Report CSTR-164, AT&T Bell Laboratories, 1993.
11. [HS91] Samuel P. Harbison and Guy L. Steele Jr. *C—A Reference Manual*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, third edition, 1991. An authoritative reference to the C programming language, and a good companion to Kernighan and Ritchie.
12. [KL93] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0*. Addison-Wesley, Reading, MA, USA, 1993. `https://ctan.org/pkg/cweb`.
13. [Knu84] Donald E. Knuth. A torture test for TEX. Report No. STAN-CS-84-1027, Stanford University, Department of Computer Science, 1984.
14. [Knu86a] Donald E. Knuth. A Torture Test for METAFONT. Report No. STAN-CS-86-1095, Stanford University, Department of Computer Science, 1986.
15. [Knu86b] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
16. [Knu86c] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
17. [Knu86d] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
18. [Knu86e] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
19. [Knu86f] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
20. [Knu89] Donald E. Knuth. The errors of TEX. *Software—Practice and Experience*, 19(7):607–681, July 1989. This is an updated version of Knuth:1988:ET.

21. [Knu90] Donald Knuth. Virtual Fonts: More Fun for Grand Wizards. *TUGboat*, 11(1):13–23, April 1990. `https://tug.org/TUGboat/tb11-1/tb27knut.pdf`.
22. [Knu92] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.
23. [Lam94] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994. Reprinted with corrections, 1996.
24. [Lia83] Franklin Mark Liang. Word hy-phen-a-tion by com-pu-ter. Technical Report STAN-CS-83-977, Stanford University, August 1983. `https://tug.org/docs/liang/liang-thesis.pdf`.
25. [Mac91] Pierre A. MacKay. Looking at the pixels: Quality control for 300 dpi laser printer fonts, especially Metafonts. In Robert A. Morris and Jacques Andre, editors, *Raster Imaging and Digital Typography II—Papers from the second RIDT meeting, held in Boston, Oct. 14–16, 1991*, pages 205–215, New York, 1991. Cambridge University Press.

Index