

TEX Live

for version 2014
October 2013

Karl Berry
Peter Breitenlohner
Norbert Preining
<http://tug.org/tex-live>

This file documents the T_EX Live system, etc.

Copyright © 2013 Karl Berry, Peter Breitenlohner, & Norbert Preining.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the T_EX Users Group.

Table of Contents

1	Introduction	1
1.1	TeX Live Mailing Lists	1
2	The TeX Live Build System	3
2.1	Requirements to Build TeX Live	3
2.2	Building and Installing TeX Live	4
2.2.1	Installation Paths	4
2.2.2	Linked Scripts	5
2.2.3	Distro Builds	5
2.2.3.1	Configuring for a Distro	5
2.3	The TeX Live Source Tree	6
2.3.1	The Top-Level Directories	6
2.3.2	Structure of Library Modules	7
2.3.2.1	The png library in libs/libpng	7
2.3.2.2	The zlib library in libs/zlib	8
2.3.2.3	The freetype library in libs/freetype2	8
2.3.2.4	The kpathsea library in texk/kpathsea	8
2.3.3	Structure of Program Modules	8
2.3.3.1	The t1utils package in utils/t1utils	8
2.3.3.2	The xindy package in utils/xindy	9
2.3.3.3	The xdvik package in texk/xdvik	9
2.3.3.4	The subdirectory utils/asymptote	9
2.3.4	Adding New Modules	9
2.3.4.1	Adding a New Program Module	9
2.3.4.2	Adding a New Generic Library Module	10
2.3.4.3	Adding a New TeX Specific Library Module	10
2.3.5	Using Libraries from the TeX Live Tree	11
2.3.5.1	configure.ac	11
2.3.5.2	Makefile.am	11
2.4	List of all Configure Options	11
2.4.1	Global Configure Options	12
2.4.1.1	--disable-native-texlive-build	12
2.4.1.2	--prefix, --exec-prefix, --bindir,	12
2.4.1.3	--enable-multiplatform	12
2.4.1.4	--enable-cxx-runtime-hack	12
2.4.1.5	--enable-libtool-hack	12
2.4.1.6	--enable-shared	12
2.4.1.7	--disable-largefile	13
2.4.1.8	--without-x	13
2.4.1.9	--enable-compiler-warnings=[no min yes max all]	13
2.4.1.10	--disable-missing	13
2.4.1.11	--enable-silent-rules	13

2.4.1.12	<code>--without-ln-s</code>	13
2.4.1.13	<code>--enable-maintainer-mode</code>	13
2.4.2	Configure Options for Program Modules	13
2.4.2.1	<code>--enable-prog</code> , <code>--disable-prog</code>	13
2.4.2.2	<code>--disable-all-pkgs</code>	13
2.4.2.3	Configure options for <code>texk/web2c</code>	14
2.4.2.4	Configure options for <code>texk/bibtex-x</code>	14
2.4.2.5	Configure options for <code>texk/dvipdfm-x</code>	15
2.4.2.6	Configure options for <code>texk/dvisvgm</code>	15
2.4.2.7	Configure options for <code>texk/xdvik</code>	15
2.4.2.8	Configure options for <code>utils/xindy</code>	15
2.4.3	Configure Options for Library Modules	15
2.4.3.1	<code>--with-system-lib</code>	15
2.4.3.2	Configure Options for <code>kpathsea</code>	16
2.4.3.3	Configure Options for <code>fontconfig</code>	16
2.4.3.4	Configure Options for system <code>poppler</code>	16
2.4.4	Interesting and/or Important Variables	16
2.4.4.1	<code>CC</code> , <code>CXX</code> , <code>CPPFLAGS</code> ,	16
2.4.4.2	<code>FT2_CONFIG</code> , <code>ICU_CONFIG</code> , <code>PKG_CONFIG</code>	16
2.4.4.3	<code>CLISP</code>	16
2.4.4.4	<code>PERL</code> , <code>LATEX</code> , <code>PDFLATEX</code>	17
2.4.4.5	<code>TL_PLATFORM</code>	17
2.4.4.6	<code>KPSEWHICH</code>	17
2.4.4.7	<code>MAKE</code> , <code>SED</code> ,	17
2.5	Cross Compilation	17
2.5.1	Configuring for Cross Compilation	17
2.5.2	Cross Compilation Problems	18
2.6	Coding Rules	19
2.6.1	Declarations	19
2.6.1.1	ANSI C function prototypes and definitions	19
2.6.1.2	Static Functions	19
2.6.1.3	Extern Functions	19
2.6.1.4	Variables	20
2.6.2	Const	20
2.6.2.1	Function Parameters	20
2.6.2.2	What Must be Avoided	20
2.6.2.3	What Should be Avoided	20

Index	21
--------------------	-----------

1 Introduction

This manual corresponds to version 2014 of the T_EX Live system, released in October 2013.

1.1 T_EX Live Mailing Lists

First, two common kinds of messages which should not go to any TeX Live list:

- Package bug reports must go to the package author. T_EX Live redistributes what is on CTAN without changes.
- General T_EX/L^AT_EX usage questions should go to one of general help resources. The T_EX Live lists are for T_EX Live topics specifically.

The following mailing lists related to T_EX Live are hosted on **tug.org**:

- **tex-live@tug.org** - bug reports, package requests, license issues, and general T_EX Live discussion of any kind.
- **tlbuild@tug.org** - specifically about building the binaries from the sources included in T_EX Live, and additional custom binaries.
- **tldistro@tug.org** - specifically about packaging T_EX Live for complete OS distributions.
- **tldoc@tug.org** - specifically about the base T_EX Live documentation and its translations.
- **tlsecurity@tug.org** - specifically for security-related reports.

You can (un)subscribe to each or peruse their archives via the web interfaces listed above.

2 The T_EX Live Build System

The T_EX Live build system has been redesigned in 2009, using Autoconf, Automake, and Libtool. Thus

```
configure && make && make check && make install
```

or the more or less equivalent top-level Build script suffice to build and install the T_EX Live programs, where **make check** performs various test of the generated programs that are not necessary but strongly recommend.

The main components of the T_EX Live build system are T_EX specific programs in subdirectories **texk/prog**, utility programs in subdirectories **utils/prog**, T_EX specific libraries in the subdirectories **texk/kpathsea** and **texk/ptexenc** used by the T_EX specific programs, and generic libraries in subdirectories **libs/lib**.

The primary design goal is modularity. Each program and library module (or package) specifies its own requirements and properties, such as required libraries, whether an installed (system) version of a library can be used, configure options to be seen at the top-level, and more. An explicit list of all available modules is only kept in one central place.

A second, related goal is to configure and build each library before configuring any other (program or library) module using that library. This allows to check for properties and features of a library built as part of the T_EX Live tree in much the same way as for a system version of that library.

All generic libraries and several programs are maintained independently. The corresponding modules use (parts of) the distributed source tree and document any modifications of that source tree.

All this should simplify upgrading of modules maintained independently and/or integrating new modules into the T_EX Live build system.

2.1 Requirements to Build T_EX Live

Building the T_EX Live programs requires

- C and C++ compilers
- GNU **make**

and uses the libraries included in the T_EX Live source tree. There are, however, some additional requirements

- **xindy** requires GNU **clisp** and in addition **perl**, **latex**, and **pdflatex** to build the rules and/or documentation.
- Autodetection of the T_EX Live platform name for **biber** requires **perl**.
- **xpdfopen** and **xdvik** require X11 headers and libraries (often a “development” package, not installed by default).
- XeTeX requires an Objective C++ compiler under Mac OS X or otherwise **libfontconfig** (again both headers and library).
- Modification of any **.y** or **.l** source files requires **bison** or **flex** to update the corresponding C sources.
- Modification of any part of the build system (**M4** macros, **configure.ac**, **Makefile.am**, or their fragments) requires GNU **M4**, GNU **Autoconf**, GNU **Automake**, and GNU **Libtool** to update the generated files.

Without the required tools modifying such files or building these programs must avoided, e.g., via `--disable-xindy` or `--without-x`.

2.2 Building and Installing TeX Live

The top-level `Build` script can be used to configure and build everything in a subdirectory (root of the build tree, default `Work`), install everything in an other subdirectory (default `inst`), and finally run `make check`. Several details of this process can be specified via environment variables and a few leading options; all remaining arguments (assignments or options) are passed to the `configure` script. Or one can run `configure` and `make` in a suitable empty subdirectory.

Running the top-level `configure` script configures just the top-level and the subdirectories `libs`, `utils`, and `texk`. Running `make` at the top-level first iterates over all TeX specific libraries, and then runs `make` in `libs`, `utils`, and `texk` to iterate over all generic libraries, utility programs, and TeX specific programs. These iterations consist of two steps:

- (1) For each library or program module not yet configured run `configure` adding the configure option `--disable-build` if the module need not be built or otherwise run `make all`.
- (2) For each library or program module that must be built run `make` for the selected target(s): `default` or `all` to (re-)build, `check` to run tests, `install` etc.

Running the top-level `make` a second time iterates again over all library and program modules and rebuild rebuild as necessary due to modified source files.

In case configuring or building a module fails, one could fix the problem, remove the subdirectory for the module from the build tree, and rerun the top-level `make` (or `Build` script with `--no-clean` as additional first argument).

With the configure option `--disable-all-pkgs` all program and library modules are configured but none of them are built. The `Makefile` for each such module contains all build rules and dependencies and can be invoked to build an individual program or library and causes to first build any required libraries. This build "on demand" procedure is used, e.g., in the `luatex` repository to build LuaTeX, essentially from a subset of the complete TeX Live tree. Similarly, when, e.g., building ε -TeX has been disabled (as by default), one can run `make etex` (or `make etex.exe`) in `texk/web2c/` to build ε -TeX (although there is no simple way to install ε -TeX).

The TeX Live build system carefully formulates dependencies as well as make rules when a tool (such as `tangle`, `ctangle`, or `convert`) creates several output files. This allows for parallel builds (`make -j n` with $n > 1$ or even `make -j`) that can considerably speed up the TeX Live build on multi core systems. Further speed up can be achieved by using a configure cache file, i.e., with the top-level configure option `-C`.

The command `make dist` (at the top-level) creates a tarball `tex-live-yyyy-mm-dd.tar.xz` from the TeX Live source tree, whereas `make dist-check` also verifies that this tarball can be used to build and install all of TeX Live.

2.2.1 Installation Paths

Running `make install` (or `make install-strip`) installs executables in `bindir`, libraries in `libdir`, headers in `includedir`, data (including "linked scripts") in `datarootdir/texmf-dist`, manpages in `mandir`, and TeX info files in `infodir`. The values of these directories are determined by `configure` and can be specified explicitly as

options such as `--prefix=prefix` or `--bindir=bindir`; otherwise they are given by their usual Autoconf defaults

<i>prefix</i>	<i>/usr/local</i>
<i>exec_prefix</i>	<i>prefix</i>
<i>bindir</i>	<i>exec_prefix/bin</i>
<i>libdir</i>	<i>exec_prefix/lib</i>
<i>includedir</i>	<i>prefix/include</i>
<i>datarootdir</i>	<i>prefix/share</i>
<i>mandir</i>	<i>datarootdir/man</i>
<i>infodir</i>	<i>datarootdir/info</i>

and modified as follows:

- If the option `--enable-multiplatform` is given (or implied for a native T_EX Live build), */host*, i.e., the canonical host name is appended to *bindir* and *libdir*.
- In a native T_EX Live build *datarootdir* is set to *prefix*, *mandir* to *prefix/texmf-dist/doc/man*, and *infodir* to *prefix/texmf-dist/doc/info*.

The top-level `configure` script displays all these installation paths.

For a native T_EX Live build either for the T_EX Live DVD or for an additional platform the contents of *bindir* should be copied to the directory *Master/bin/arch* of the T_EX Live tree where *arch* is the T_EX Live platform name corresponding to canonical host name *host*, the contents of *libdir* and *includedir* can be discarded, and everything else should match files already present in the T_EX Live tree.

2.2.2 Linked Scripts

Quite a few executables are architecture independent Shell, Perl, or other scripts. Some of them are maintained as part of the T_EX Live source tree, but most are maintained elsewhere with copies under *texk/texlive/linked_scripts*. They are installed under *datarootdir/texmf-dist/scripts*; for Unix-like systems there is a symbolic link pointing, e.g., from *bindir/ps2eps* to *datarootdir/texmf-dist/scripts/ps2eps/ps2eps.pl* whereas for Windows *bindir/ps2eps.exe* is a copy of a small standard binary serving the same purpose. One reason for all this is to avoid having many copies for the same script, but most importantly this allows to invoke the same Perl or other script under Windows.

2.2.3 Distro Builds

Although they use the same code base, building for a T_EX Live binary distribution as shipped by the user groups may be quite different from a 'distro' build for, e.g., some kind of Linux distribution, a *Bsd or Mac OS X port, or similar.

While a T_EX Live binary distribution uses shared libraries (*libc*, *libm*, X11 libraries, and *libfontconfig*) only when absolutely necessary, a distro might use as many shared libraries as possible, including T_EX specific libraries such as *libkpathsea*. In addition the installation paths will, in general, be quite different.

2.2.3.1 Configuring for a Distro

For a distro build you must use

`--disable-native-texlive-build`

and should use

```
--with-banner-add=/SomeDistro
```

to identify your distro. You may specify

```
--enable-shared
```

to build shared TeX specific libraries and might add

```
--disable-static
```

to not build the static ones. You would like to use

```
--with-system-lib
```

for as many libraries as possible and may then have to add `--with-lib-includes=dir` and/or `--with-lib-libdir=dir`.

You should specify

```
--prefix=/usr
```

or perhaps `--prefix=/opt/TeXLive` and may have to add

```
--libdir=\${exec_prefix}/lib64
```

for 64bit bi-arch (Linux) systems.

To make a usable TeX installation, you need (thousands of) support files in addition to the binaries that are built and installed here. The support files are maintained completely independently. The best basis for dealing with them is the TeX Live (plain text) database and/or the TeX Live installer.

2.3 The TeX Live Source Tree

The TeX Live source tree is the subtree rooted at `Build/source` of the complete TeX Live tree and contains the sources for all TeX Live executables as well as `make` rules to build and install them together with some of their support files.

2.3.1 The Top-Level Directories

In general, the TeX Live build system uses the latest released versions of the GNU build tools, installed directly from the original GNU releases (e.g., by building them with

```
configure --prefix=/usr/local/gnu
```

and having `PATH` start with `/usr/local/gnu/bin`). Currently these are

```
autoconf (GNU Autoconf) 2.69
automake (GNU automake) 1.14
ltmain.sh (GNU libtool) 2.4.2
bison (GNU Bison) 3.0
flex 2.5.37
m4 (GNU M4) 1.4.17
makeinfo (GNU texinfo) 5.2
```

These versions should be used to update the generated files (e.g., `configure` or `Makefile.in`) in all or parts of the TeX Live tree when some of their sources have been changed. This can be done explicitly with the top-level `reautoconf` script or implicitly by using the configure option `--enable-maintainer-mode`.

The files in the SVN repository are all up to date, but this need not be reflected by their timestamps. In order not to run `bison`, `flex`, or `makeinfo` it may be necessary to `touch` the generated (`.c`, `.h`, or `.info`) files. With `--enable-maintainer-mode` mode it may also be necessary to `touch` first `aclocal.m4`, then `configure` and `config.h.in` (or `c-auto.in`), and finally all `Makefile.in` files.

The files `config.guess`, `config.sub`, etc. for most packages are kept centrally in `build-aux/`, sourced from GNU Gnulib (<http://www.gnu.org/software/gnulib>), which in turn pulls them from their ultimate upstream source repository. There are, however, independent copies in, e.g., `libs/freetype2/freetype-*/builds/unix/`, and similar places. The `reautoconf` script does not take care of those, but a T_EX Live cron job keeps them in sync (nightly).

The subdirectories `m4` and `am` contain M4 macros and `Makefile.am` fragments respectively, all of them used in many places.

The file `m4/kpse-pkgs.m4` contains lists of all program and library modules; missing modules are, however, silently ignored. Each such module contributes fragments defining its capabilities and requirements to the `configure.ac` scripts at the top-level and in the subdirectories `libs`, `utils`, and `texk`. The fragments from program modules supply `configure` options to disable or enable building them, those from library modules specify if an installed (system) version of that library can be used. This decides which modules need to be built, although all modules must be configured for the benefit of `make` targets such as `dist` or `distcheck`.

The subdirectory `extra` contains things which are not part of the TeX Live build, but are present for convenience, e.g., `epstopdf` developed here or `xz` required by the T_EX Live installer.

2.3.2 Structure of Library Modules

The structure of library modules is best explained with a few examples.

2.3.2.1 The png library in `libs/libpng`

This generic library uses the distributed source tree in, e.g., `libpng-1.5.17` with all modifications recorded in `libpng-1.5.17-PATCHES/*`. The `configure.ac` fragment `ac/withenable.ac` contains

```
KPSE_WITH_LIB([libpng], [zlib])
```

with the module name and indicating the dependency on `zlib`. A third argument `tree` would specify that the library from the T_EX Live tree can not be replaced by a system version. That not being the case, a second fragment `ac/libpng.ac` contains

```
KPSE_TRY_LIB([libpng],
             [#include <png.h>],
             [png_structp png; png_voidp io; png_rw_ptr fn;
             png_set_read_fn(png, io, fn);])
```

and provides the simple C code

```
#include <png.h>
int main ()
{ png_structp png; png_voidp io; png_rw_ptr fn;
  png_set_read_fn(png, io, fn);
  return 0; }
```

used to verify the usability of a system version. The similar macro `KPSE_TRY_LIBXX` would provide some C++ code. These fragments are included by `configure.ac` at the top levels.

A proxy build system (`configure.ac`, `Makefile.am`, and `include/Makefile.am`) ignores the distributed one and consequently a few generated files and auxiliary scripts are

removed from the distributed source tree. The public headers `png.h`, `pngconf.h`, and `pnglibconf.h` are 'installed' (as symlinks) under `include/` in the build tree exactly as they are for a system version under, e.g., `usr/include/`.

The module is supplemented by the file `m4/kpse-libpng-flags.m4` that defines the M4 macro `KPSE_LIBPNG_FLAGS` used by all modules depending on this library in their `configure.ac` to generate the make variables `LIBPNG_INCLUDES` for use in `CPPFLAGS`, `LIBPNG_LIBS` for use in `LDADD`, `LIBPNG_DEPEND` for use as dependency, and `LIBPNG_RULE` defining make rules to rebuild the library.

In addition `m4/kpse-libpng-flags.m4` supplies the configure option `--with-system-libpng` and uses `pkg-config` to determine the flags required for the system library.

2.3.2.2 The zlib library in `libs/zlib`

This generic library is quite analogous to `libpng` but does not depend on any other library. The file `m4/kpse-zlib-flags.m4` supplies the configure option `--with-system-zlib` as well as `--with-zlib-includes` and `--with-zlib-libdir` to specify non standard locations of the `zlib` headers and/or library.

2.3.2.3 The freetype library in `libs/freetype2`

This module uses a wrapper build system with an almost trivial `configure.ac` and with `Makefile.am` that invokes `configure` and `make` for the distributed source, followed by `make install` with the build tree as destination. The flags required for the system library are obtained through `freetype-config`.

2.3.2.4 The kpathsea library in `texk/kpathsea`

This is one of the TeX specific libraries that are maintained as part of TeX Live. Other than the generic libraries they are (static and/or shared) Libtool libraries and are installed together with the programs. They are, however, not part of the TeX Live DVD as distributed by TeX user groups.

It is possible, although quite unusual to specify the configure option `--with-system-kpathsea` in order to use a system version of the library and it may then be necessary to specify `--with-kpathsea-includes` and/or `--with-kpathsea-libdir`.

In addition to `ac/withenable.ac` and `ac/kpathsea.ac` there is a third fragment `ac/mktex.ac` included by both `ac/withenable.ac` and `configure.ac` that supplies configure options such as `--enable-mktextfm-default` determining the compile time default whether or not to run `mktextfm` to generate a missing `.tfm` file. Note, however, that the command line options `-mktex=tfm` or `-no-mktex=tfm` for TeX-like engines override this default.

2.3.3 Structure of Program Modules

The structure of program modules is again best explained with a few examples.

2.3.3.1 The t1utils package in `utils/t1utils`

Once again we use the distributed source tree `t1utils-1.37` with modifications documented in `t1utils-1.37-PATCHES/*` and a proxy build system consisting of `configure.ac` and `Makefile.am`. The fragment `ac/withenable.ac` contains

```
KPSE_ENABLE_PROG([t1utils])
```

specifying the module name without any dependencies and supplies the configure option `--disable-t1utils`.

2.3.3.2 The xindy package in utils/xindy

This module uses the distributed source tree `xindy-2.4` with modifications documented in `xindy-2.4-PATCHES/*`, a proxy `configure.ac`, and a wrapper `Makefile.am` that descends into `xindy-2.4`. This requires that the distributed Makefiles allow a VPATH build, can handle all targets, and do not refer to `${top_srcdir}` or `${top_builddir}`. The fragment `ac/withenable.ac` contains

```
KPSE_ENABLE_PROG([xindy], , [disable native])
m4_include(kpse_TL[utils/xindy/ac/xindy.ac])
m4_include(kpse_TL[utils/xindy/ac/clisp.ac])
```

where `disable` in the third argument indicates that `xindy` is only built if explicitly enabled by `--enable-xindy` (because it requires `clisp`) and `native` disallows cross compilation. The additional fragments `ac/xindy.ac` and `ac/clisp.ac` specify more configure options to be seen at the top-level with `ac/xindy.ac` also included by `configure.ac`.

2.3.3.3 The xdvik package in texk/xdvik

This package is maintained as part of the T_EX Live tree with sources in its top-level and the subdirectory `gui`. The fragment `ac/withenable.ac` contains

```
dn1 extra_dirs = texk/xdvik/squeeze
KPSE_ENABLE_PROG([xdvik], [kpathsea freetype2], [x])
m4_include(kpse_TL[texk/xdvik/ac/xdvik.ac])
```

and specifies the dependency on the `kpathsea`, `freetype`, and X11 libraries. The M4 comment (following `dn1`) signals the subsidiary `squeeze/configure.ac`. This is needed because the main executable `xdvi-bin` (to be installed as, e.g., `xdvi-xaw`) is for the `host` system whereas the auxiliary program `squeeze/squeeze` has to run on the `build` system and in a cross compilation they differ. The additional fragment `ac/xdvik.ac` is also included by `configure.ac` and supplies the configure option `--with-xdvi-x-toolkit` also seen at the top-level.

2.3.3.4 The subdirectory utils/asyptote

This subdirectory contains the sources for `asy` and `xasy` but due to its complexity and prerequisites (e.g., `libGL`) it is not part the T_EX Live build system. These programs must be built and installed independently, but are contained in the T_EX Live DVD together with their support files.

2.3.4 Adding New Modules

2.3.4.1 Adding a New Program Module

A T_EX specific program module in a subdirectory `texk/prog` may use the T_EX specific libraries and is included by adding its name `prog` to the M4 list `kpse_texk_pkgs` defined in `m4/kpse-pkgs.m4`. A generic program module in a subdirectory `utils/prog` must not use the T_EX specific libraries and is included by adding its name `prog` to the M4 list

`kpse_utils_pkgs` in `m4/kpse-pkgs.m4`. Apart from the program sources and build system (`configure.ac` and `Makefile.am`) the subdirectory `texk/prog` or `utils/prog` must provide a fragment `ac/withenable.ac` that contains the M4 macro `KPSE_ENABLE_PROG` defined in `m4/kpse-setup.m4` with `prog` as mandatory first argument and three optional arguments: a list of required libraries from the TeX Live tree, a list of options (`disable` if this module is not to be built without the configure option `--enable-prog`, `native` if cross compilation is not possible, `x` if the program requires X11 libraries), and a comment added to the help text for the configure option `--enable-prog` or `--disable-prog`.

If the module requires specific configure options to be seen at the top-level, they should be defined in an additional fragment `ac/prog.ac` included from `ac/withenable.ac` and `configure.ac`.

2.3.4.2 Adding a New Generic Library Module

A generic library module in a subdirectory `libs/lib` must not depend on TeX specific libraries and is included by adding its name `lib` to the M4 list `kpse_libs_pkgs` in `m4/kpse-pkgs.m4` (before other libraries from the TeX Live tree it depends on). As for program modules the subdirectory `libs/lib` must contain the sources and build system for the library (and any installable support programs) and a fragment `ac/withenable.ac` that contains the M4 macro `KPSE_WITH_LIB` defined in `m4/kpse-setup.m4` with `lib` as mandatory first argument and two optional arguments: a list of required libraries from the TeX Live tree, and a list of options (currently only `tree` if this library can not be replaced by a system version).

If a system version can be used, a second fragment `ac/lib.ac` containing the M4 macro `KPSE_TRY_LIB` (or `KPSE_TRY_LIBXX`) with `lib` as mandatory first argument and two additional arguments for the Autoconf macro `AC_LANG_PROGRAM` used to compile and link a small C (or C++) program as sanity check for using the system library.

In addition a file `m4/kpse-lib-flags` (at the top-level) must define the M4 macro `KPSE_LIB_FLAGS` (all uppercase) setting up the make variables `LIB_INCLUDES`, `LIB_LIBS`, `LIB_DEPEND`, and `LIB_RULE` with the values required for `CPPFLAGS`, `LDADD`, dependencies, and a (multiline) make rule to rebuild the library when necessary, all that for the library from the TeX Live tree or perhaps for a system version.

If a system library is allowed `KPSE_LIB_FLAGS` also provides the configure option `--with-system-lib` and uses the additional M4 macro `KPSE_LIB_SYSTEM_FLAGS` to generate the make variables for a system library. Furthermore the definition of the M4 macro `KPSE_ALL_SYSTEM_FLAGS` in `m4/kpse-pkgs.m4` must be extended by the line

```
AC_REQUIRE([KPSE_LIB_SYSTEM_FLAGS])
```

2.3.4.3 Adding a New TeX Specific Library Module

A TeX specific library module in a subdirectory `texk/lib` may depend on other TeX specific libraries but must not depend on any generic library from the TeX Live tree. It is included as is a generic library module with these modifications:

- The library name `lib` is added to the M4 list `kpse_texlibs_pkgs` also in `m4/kpse-pkgs.m4`.
- The fragment `ac/withenable.ac` must use `KPSE_WITH_TEXLIB`.

2.3.5 Using Libraries from the T_EX Live Tree

The T_EX Live build system provides Autoconf macros for all libraries in the T_EX Live tree. A program requiring additional libraries must use standard Autoconf and Automake features (such as `AC_ARG_WITH` for search paths, `AC_CHECK_LIB` and `AC_CHECK_FUNCS` for library functions, or `AC_CHECK_HEADERS` for header files).

2.3.5.1 `configure.ac`

In order to use, e.g., `libpng` (either an installed version or from the T_EX Live tree) `configure.ac` must use

```

KPSE_ZLIB_FLAGS
KPSE_LIBPNG_FLAGS

```

to define the Make variables `LIBPNG_DEPEND`, `LIBPNG_INCLUDES`, `LIBPNG_LIBS`, and `LIBPNG_RULE` (and analogous `ZLIB_*` variables). To examine some `libpng` features, `configure.ac` must use

```

KPSE_ADD_FLAGS([zlib])

```

followed by tests for `zlib` features (if any) and

```

KPSE_ADD_FLAGS([libpng])

```

followed by tests for `libpng` features. These macros temporarily add appropriate values to `CPPFLAGS` and `LIBS`. The tests must be terminated by

```

KPSE_RESTORE_FLAGS

```

to restore `CPPFLAGS` and `LIBS` to their former values.

2.3.5.2 `Makefile.am`

In order to use, e.g., `libpng` (and `zlib`) `Makefile.am` must add

```

${LIBPNG_INCLUDES} ${ZLIB_INCLUDES}

```

to `AM_CPPFLAGS` (or target specific `target_CPPFLAGS`) and

```

${LIBPNG_LIBS} ${ZLIB_LIBS}

```

to `LDADD` or `target_LDADD`, and should add

```

${LIBPNG_DEPEND} ${ZLIB_DEPEND}

```

to `target_DEPENDENCIES` such that rebuilding `zlib` and/or `libpng` causes the target to be rebuilt. Finally the two lines

```

@ZLIB_RULE@
@LIBPNG_RULE@

```

create the rules required to (re-)build `zlib` and `libpng` when necessary.

2.4 List of all Configure Options

Corresponding to the large number of program and library modules there are plenty of configure options, most of which are described here. The command

```

configure --help

```

at the top-level gives an exhaustive list of all global options and a few important module specific ones, whereas, e.g.,

```

texk/lcdf-typetools/configure --help

```

also displays the `lcdf-typetools` specific options not shown at the top-level. The help text also mentions several influential environment variables, but for T_EX Live it is better to specify them as assignments on the command line.

The `./Build` script used to make the binaries shipped with TeX Live invokes the top-level `configure` with a few additional options. Any defaults discussed below are those for the base `configure` script; invoking `configure` via `./Build` may yield different results.

Defaults for most options are set at the top-level and propagated explicitly to all subdirectories. Options specified on the command line are checked for consistency but are never modified.

2.4.1 Global Configure Options

2.4.1.1 `--disable-native-texlive-build`

If enabled (the default), build for a TeX Live binary distribution as shipped by the user groups; this requires GNU `make` and implies `--enable-multiplatform` and `--enable-cxx-runtime-hack` unless they are explicitly disabled and enforces `--disable-shared`.

If building TeX Live for a GNU/Linux or other distribution, this would be disabled and system versions of most libraries would be used (see below). This may require GNU `make`, but will also try without it.

A related option `--enable-texlive-build` is automatically passed to all subdirectories (and can not be disabled). Subdirectories that could also be built independently from the TeX Live tree (such as `utils/xindy` or `texk/dvipng`) can use this option, e.g., to choose TeX Live specific installation directories.

2.4.1.2 `--prefix`, `--exec-prefix`, `--bindir`, ...

These options specify various installation directories as usual, all of them still prefixed by the value of an assignment for `DESTDIR` on the `make` command line (see [Section “Installation in a temporary location” in GNU Automake](#)).

2.4.1.3 `--enable-multiplatform`

If enabled, install executables and libraries in platform dependent subdirectories of `EPREFIX/bin` and `EPREFIX/lib` (unless `--bindir=dir` or `--libdir=dir` is specified), where `EPREFIX` is the value given or implied for `exec_prefix`. The values for `bindir` and `libdir` are automatically propagated to all subdirectories.

2.4.1.4 `--enable-cxx-runtime-hack`

If enabled and when using `g++`, try to statically link with `libstdc++`, somewhat improving portability.

2.4.1.5 `--enable-libtool-hack`

If enabled (at present the default for all platforms), prevents `libtool` from linking explicitly with `dependency_libs`.

2.4.1.6 `--enable-shared`

Build shared versions of the TeX specific (`kpathsea` and `ptexenc`) libraries; this is not allowed for a native TeX Live build.

2.4.1.7 --disable-largefile

Omit large file support (LFS), needed on most 32-bit Unix systems for files with 2GB or more. The size of DVI and GF files must always be < 2GB. With LFS there should be no limit on the size of PDF files created by `pdftex` or PS files created by `dvips`. The size of PDF images included by `pdftex` must, however, be < 4GB when using `xpdf` and < 2GB when using older versions of `poppler` (even on 64-bit systems with LFS), whereas `poppler` Version 0.23 and later imposes no such limit.

2.4.1.8 --without-x

Disable all programs using the X Window System.

2.4.1.9 --enable-compiler-warnings=[no|min|yes|max|all]

Enable various degrees of compiler warnings for (Objective) C and C++. The default is `yes` in maintainer-mode and `min` otherwise. This option defines `WARNING_[OBJ]C[XX]FLAGS` but these flags are not used in all library and program modules. Using them should help to resolve portability problems.

At the moment these warning flags are only defined for the GNU compilers but flags for other compilers could be added when needed.

2.4.1.10 --disable-missing

Immediately terminate the build process if a requested program or feature must be disabled, e.g., due to missing libraries.

2.4.1.11 --enable-silent-rules

Enable the use of less verbose build rules. When using GNU `make` or another `make` implementation that supports nested variable expansions you can always specify `V=1` on the `make` command line to get more respectively `V=0` to get less verbosity.

2.4.1.12 --without-ln-s

Required to build without working `ln -s` for a Unix-like system. But note that `make install` will not create anything useful and might even fail.

2.4.1.13 --enable-maintainer-mode

Enable `make` rules and dependencies not useful (and sometimes confusing) to the casual user. This requires current versions of the GNU build tools.

2.4.2 Configure Options for Program Modules

2.4.2.1 --enable-prog, --disable-prog

Do or do not build and install the program(s) of the module *prog*.

2.4.2.2 --disable-all-pkgs

Do not build any program modules, except those explicitly enabled. Without this option, all modules are built except those that are explicitly disabled or specify `disable` in their `ac/witenable.ac` fragment.

2.4.2.3 Configure options for `texk/web2c`

`--with-banner-add=str`

Add *str* to the default version string (TeX Live year or Web2C year) appended to banner lines. This is ignored for a native TeX Live build, but distro builds should specify, e.g., */SomeDistro*.

`--with-editor=cmd`

Specify the command *cmd* to invoke from the *e* option of TeX-like engines, replacing the default `vi +%d '%s'` for Unix or `texworks --position=%d "%s"` for Windows.

`--enable-auto-core`

This option causes TeX & METAFONT to produce a core dump when a particular hacky filename is encountered.

`--disable-dump-share`

Make the `fmt/base` dump files architecture dependent (somewhat faster on little-endian architectures).

`--disable-ipc`

Disable TeX's `--ipc` option.

`--disable-tex, --enable-etex, ...`

Do not or do build the various TeX, METAFONT, and MetaPost engines (defaults are defined in the fragment `texk/web2c/ac/web2c.ac`).

`--enable-tex-synctex, --disable-etex-synctex, ...`

Build the various TeX-like engines with or without SyncTeX support (ignored for a native TeX Live build, defaults are again defined in `texk/web2c/ac/web2c.ac`).

`--with-mf-x-toolkit`

Use the X toolkit (`libXt`) for METAFONT (default is yes).

`--enable-*win`

Include various types of other window support for METAFONT.

`--disable-mf-nowin`

Do not build a separate non-graphically-capable METAFONT.

`--disable-web-progs`

Do not build the WEB programs `bibtex`, ..., `weave`, e.g., if you just want to (re)build some engines.

`--disable-omfonts`

Build the WEB versions of the Omega font utilities `ofm2opl`, `opl2ofm`, `ovf2ovp`, and `ovp2ovf` instead of the C version `omfonts`. The WEB and C versions should be roughly equivalent.

2.4.2.4 Configure options for `texk/bibtex-x`

The former modules `bibtex8` and `bibtexu` have been merged into `bibtex-x` (extended BibTeX).

`--disable-bibtex8`

Do not build the `bibtex8` program.

`--disable-bibtexu`

Do not build the `bibtexu` program (building `bibtexu` requires ICU libraries).

2.4.2.5 Configure options for texk/dvipdfm-x

The former modules `dvipdfmx` and `xdvipdfmx` have been merged into `dvipdfm-x`.

`--disable-dvipdfmx`

Do not build the `dvipdfmx` program.

`--disable-xdvipdfmx`

Do not build the `xdvipdfmx` program (building `xdvipdfmx` requires the `freetype` library).

2.4.2.6 Configure options for texk/dvisvgm

`--with-system-libgs`

Build `dvisvgm` using installed Ghostscript (`gs`) headers and library (not allowed for a native T_EX Live build). The default is to load the `gs` library at runtime if possible, or otherwise disable support for PostScript specials.

`--with-libgs-includes=dir`, `--with-libgs-libdir=dir`

Specify non standard locations of the Ghostscript headers and library.

2.4.2.7 Configure options for texk/xdvik

`--with-gs=path`

Hardwire the location of Ghostscript (`gs`).

`--with-xdvi-x-toolkit=kit`

Use toolkit `kit` (`motif`/`xaw`/`xaw3d`/`neXtaw`) for `xdvi`. The default is `motif` if available, else `xaw`.

2.4.2.8 Configure options for utils/xindy

`--enable-xindy-rules`

Build and install xindy rules (default: yes, except for a native T_EX Live build).

`--enable-xindy-docs`

Build and install xindy documentation (default: yes, except for a native T_EX Live build).

`--with-clisp-runtime=path`

Specifies the full path of the CLISP runtime (`lisp.run` or `lisp.exe`) to be installed. When specified as `default` (the default for a native TeX Live build) the path is determined by the CLISP executable; the value `system` (not allowed for a native T_EX Live build, but the default for a non-native one) indicates that `xindy` will use the installed version of `clisp` (that must be identical to the one used to build `xindy`).

`--with-recode`

Use `recode` instead of `iconv` to build the xindy rules and documentation, required for some systems where `iconv` is missing or broken.

2.4.3 Configure Options for Library Modules

2.4.3.1 `--with-system-lib`

Use an installed (system) version of the library `lib`; this option exists for most libraries, but is not allowed for a native T_EX Live build. Using a system version implies to also use system versions of all libraries (if any) this library depends on.

For many libraries there are in addition `--with-lib-includes=dir` and `--with-lib-libdir=dir` to indicate non standard search locations, others use `pkg-config` or similar to determine the required flags.

The top-level `configure` script performs a consistency check for all required system libraries and bails out if some of these tests fail.

2.4.3.2 Configure Options for kpathsea

`--enable-cmd-default`, `--disable-cmd-default`

Determine the compile time default whether or not to run `cmd=mktexmf`, `mktexpk`, `mktextfm`, `mkocp`, `mkofm`, `mktextfmt`, or `mktextex` to generate a missing MF source, PK font, TFM file, OCP file, OFM file, format file, or T_EX source respectively.

2.4.3.3 Configure Options for fontconfig

Building XeTeX on non-Mac systems requires `fontconfig` headers and library.

`--with-fontconfig-includes=dir`, `--with-fontconfig-libdir=dir`

If one or both of these options are given, the required flags are derived from them; otherwise, they are determined via `pkg-config` (if present).

2.4.3.4 Configure Options for system poppler

Building LuaTeX and XeTeX requires `poppler` either from the T_EX Live tree or system headers and library; `pdfTeX` requires either `xpdf` from the T_EX Live tree or system `poppler` headers and library.

`--with-system-poppler`

Use a system version (0.18 or better) of `poppler` for LuaTeX and XeTeX (and `pkg-config` to obtain the required flags).

`--with-system-xpdf`

Use a system version (0.12 or better) of `poppler` (and `pkg-config`) for `pdfTeX` instead of `xpdf` from the T_EX Live tree.

REFER to `--disable-largefile`

2.4.4 Interesting and/or Important Variables

The values for these variables can come either from the environment or from `configure` arguments of the form `VAR=value`.

2.4.4.1 CC, CXX, CPPFLAGS, . . .

As usual, these variables specify the name (or full path) of compilers, preprocessor flags, and similar.

2.4.4.2 FT2_CONFIG, ICU_CONFIG, PKG_CONFIG

These specify the name (or path) for the `freetype-config`, `icu-config`, and `pkg-config` commands used to determine the flags required for system versions of `libfreetype`, the ICU libraries, or many other libraries.

2.4.4.3 CLISP

Name (or path) of the `clisp` executable, used to build `xindy`.

2.4.4.4 PERL, LATEX, PDFLATEX

Name (or path) for the `perl`, `latex`, and `pdflatex` commands used, e.g., to build the `xindy` documentation

2.4.4.5 TL_PLATFORM

The utility program `biber` consists of many components bundled by the `Par::Packer` mechanism of `perl`. Therefore they are not built as part of T_EX Live but by the `biber` maintainers and third-party builders.

In order that `make install` creates all executables in `bindir`, the `biber` module contains pre-made binaries for the platforms for which they are available. T_EX Live merely checks if an executable is present for the current platform, and if so, installs it.

The complication is hidden in the phrase "current platform". T_EX Live has its own ideas about platform names, and the mapping from the canonical system name determined by `config.guess` or `config.sub` to the T_EX Live platform name is not trivial. A value given for `TL_PLATFORM` is used as T_EX Live platform name. Otherwise, we use copies of the standard `perl` modules `TeXLive/TLUtils.pm` and `TeXLive/TLConfig.pm`, to avoid duplicating the platform-detection logic.

2.4.4.6 KPSEWHICH

Name (or path) of an installed `kpsewhich` binary, used by `make check` to determine the location of, e.g., `cmbx10.tfm`.

2.4.4.7 MAKE, SED, ...

Name (or path) of GNU `make`, GNU `sed`, and similar; used at the top-level and propagated to all subdirectories.

2.5 Cross Compilation

In a cross compilation a `build` system is used to create binaries to be executed on a `host` system with different hardware and/or operating system.

In simple cases the build system can execute binaries for the host system. This typically occurs for bi-arch systems where, e.g., `i386-linux` binaries can run on `x86_64-linux` systems and `win32` binaries can run on `win64` systems. Although sometimes called "native cross", this is technically no cross compilation. In most such cases it suffices to specify suitable compiler flags. It might be useful to add the configure option `--build=host` to get the correct canonical host name, but note that this should not be `--host=host` (see [Section "Hosts and Cross-Compilation" in *Autoconf*](#))!

2.5.1 Configuring for Cross Compilation

In a genuine cross compilation binaries for the host system can not execute on the build system and it is necessary to specify the configure options `--host=host` and `--build=build` with two different values. Building binaries requires suitable "cross" tools, e.g., compiler, linker, and archiver, and perhaps a "cross" version of `pkg-config` and similar to locate host system libraries. `Autoconf` expects that these cross tools are given by their usual variables or found under their usual name prefixed with `host-`. Here a list of such tools and corresponding variables:

<code>ar</code>	<code>AR</code>
<code>freetype-config</code>	<code>FT2_CONFIG</code>
<code>g++</code>	<code>CXX</code>
<code>gcc</code>	<code>CC</code>
<code>icu-config</code>	<code>ICU_CONFIG</code>
<code>objdump</code>	<code>OBJDUMP</code>
<code>pkg-config</code>	<code>PKG_CONFIG</code>
<code>ranlib</code>	<code>RANLIB</code>
<code>strip</code>	<code>STRIP</code>

In order to, e.g., build `mingw32` binaries on `x86_64-linux` with a cross compiler found as `i386-pc-mingw32-gcc` one would specify

```
--host=i386-pc-mingw32 --build=x86_64-linux-gnu
```

or perhaps

```
--host=mingw32 --build=x86_64-linux CC=i386-pc-mingw32-gcc
```

but this might require to add `CXX` and others.

Configure arguments such as `CFLAGS=...` refer to the cross compiler. If necessary, you can specify compilers and flags for the few auxiliary C and C++ programs required for the build process as configure arguments

```
BUILDCC=...
BUILDCPPFLAGS=...
BUILDCFLAGS=...
BUILDCXX=...
BUILDCXXFLAGS=...
BUILDLDFLAGS=...
```

2.5.2 Cross Compilation Problems

The fact that binaries for the host system can not be executed on the build system causes some problems.

One problem is, that configure tests using `AC_RUN_IFELSE` can compile and link the test program but can not execute it. Such tests should be avoided if possible and otherwise must supply a pessimistic test result.

An other problem arises if the build process must execute some (auxiliary or installable) programs. Auxiliary programs can be placed into a subdirectory that is configured natively as is done for `texk/dvipsk/squeeze`, `texk/web2c/web2c`, and `texk/xdvik/squeeze`. The module `libs/freetype` uses the value of `CC_BUILD`, `build-gcc`, `gcc`, or `cc` as compiler for the auxiliary program.

The situation for installable programs needed by the build process is somewhat different. A quite expensive possibility, chosen for the `ICU` libraries in module `libs/icu`, is to first compile natively for the build system and in a second step to use these (uninstalled) programs during the cross compilation. This would also be possible for the tools such as `tangle` used in the module `texk/web2c` to build the `WEB` programs, but that would require to first build a native `kpathsea` library. To avoid this complication, cross compilation of the `WEB` or `CWEB` programs in this module requires sufficiently recent installed versions of `tangle`, `ctangle`, `otangle`, and `tie`.

Building `xindy` requires to run the host system `clisp` binary, thus cross compilation is not possible.

2.6 Coding Rules

Ideally, building all of T_EX Live with `--enable-compiler-warnings=max` should produce no (GCC) compiler warnings at all. In spite of considerable efforts into that direction we are still far from that goal and there are reasons that we may never fully reach it. Below are some rules about declarations of functions or variables and the use of `const`. These rules should be applied to all parts of the T_EX Live tree, except some of those maintained independently.

2.6.1 Declarations

2.6.1.1 ANSI C function prototypes and definitions

The T_EX Live build system no longer supports pre-ANSI C compilers. Thus all function prototypes and definitions must conform to the ANSI C standard (including `void` in the declaration of C functions with no parameters). On the other hand T_EX Live is built for many different systems, some of them not supporting the C99 standard. Therefore using C99 features should be avoided if that can easily be done. In particular C code must not contain declarations after statements or C++ type comments.

If some C99 (or later) constructs have to be used, the module should verify that they are available and otherwise provide an alternative. The module `texk/chktex` uses, e.g., the C99 function `strcpy()` that may or may not be available on a particular system, uses `AC_CHECK_DECLS([strcpy])` in `configure.ac` to test this, and provides the perhaps slightly less efficient alternative

```
#if !(defined HAVE_DECL_STPCPY && HAVE_DECL_STPCPY)
static inline char * strcpy(char *dest, const char *src)
{
    return strcpy(dest, src) + strlen(src);
}
#endif
```

in the file `Utility.h`.

2.6.1.2 Static Functions

Functions used in only one file should be declared `static`; they require no prototype except as forward declaration.

2.6.1.3 Extern Functions

Functions not declared `static`, usually because they are used in several files, require an (`extern`) prototype in exactly one header which is included in the file defining the function and in all files using that function — this is the only way to guarantee consistency between definition and use of functions. There must be no `extern` declarations sprinkled throughout the C code (with or without comment where that function is defined).

2.6.1.4 Variables

The declaration of global variables must follow analogous rules, they are either declared **static** if used in only one files or declared **extern** in exactly one header and instantiated in exactly one file.

2.6.2 Const

2.6.2.1 Function Parameters

Ideally, a function parameter not modified by the function should be declared as **const**. This is important in particular for strings (**char***) because the actual arguments are often string literals. It is perfectly legitimate and safe to use a type **char*** value for a type **const char*** variable (in an assignment, as initializer, as function argument, or as return value). It is equally safe to use a type **char**** value for a type **const char*const*** variable, but not for a type **const char**** variable since that might cause modification of a quantity supposed to be constant. Getting all **const** qualifiers right is often quite involved but can be done in most cases. There are, however, a few notable exceptions: the X11 headers are full of declarations that ought to use **const** but do not and the same is true to some extent for **libfreetype** (but not anymore for **zlib**).

2.6.2.2 What Must be Avoided

The GCC compiler warnings "assignment discards qualifiers..." and analogous warnings for "initialization", "passing arg", or "return" must be avoided under all circumstances, except when caused by X11 headers/macros or third party code.

2.6.2.3 What Should be Avoided

A type cast, e.g., from **const char*** to **char*** does not solve any problems; depending on warning options, it may only hide them. Therefore such casts should be avoided whenever possible and otherwise must be carefully analyzed to make sure that they can not cause the modification of quantities supposed to be constant.

Index

B

build system 3

F

fundamental purpose of T_EX Live 1

I

introduction 1

T

the T_EX Live build system 3

