fontscripts

Clea F. Rees*

v0.3 (SVN Rev: 10843) 2025/02/22

Abstract

fontscripts provides variant font encodings, support metrics and Lua script fragments to automate the creation of TEX/LATEX 2ε font files for 8-bit engines using l3build. A template-based system enables the automatic generation of font tables and l3build tests. Variables support the addition of encoding subset declarations for the Text Companion (`TS1`) encoding and variable scaling factors in font definition files.

The script fragments[1] make it possible to automate the generation of TEX font metrics, virtual fonts, map files etc. and the conversion of fonts and encodings. For tools which do not otherwise support it, such as fontinst, the scripts enable the automatic addition of variable scaling in font definition files. A semi-automatic system tries to ensure font encoding names are unique.

The script fragments were originally designed for fontinst, but support for `afm2tfm` (intended for fonts consisting entirely of ornaments etc.) is provided and a modular approach enables easy modification for other tools. The default configuration is intended to be cross-platform and requires only tools included in TEX Live, but the documentation includes a simple adaption for integration with FontForge and GNU make.

Contents

1 Quick Start **2**

2 Lua Script Fragments **3**

2.1 l3build Targets 3

2.2 Lua Functions 3

2.2.1 New Functions 3

2.2.2 Redefined Functions 6

2.3 Variables 8

3 Configuration **13**

*Bug tracker: codeberg.org/cfr/nfssext/issues | Code: codeberg.org/cfr/nfssext | Mirror: github.com/cfr42/nfssext

[1]The main script fragments are written in *extremely elementary* Lua. This is the first thing I've ever attempted in Lua and I am not a programmer.

4 Templates **14**
4.1 Font Tables 14
4.2 Font Tests 14

5 Customisation **15**
5.1 Examples 16
5.2 Templates 17
5.2.1 Font Tables Template 17
5.2.2 Test Template 17
5.3 Regression Tests 18

Change History **21**

Index **22**

1 Quick Start

This assumes you are familiar with basic usage of l3build. If not, please see l3build's manual for basic setup.

Configuring your project for l3build includes creation of a file, build.lua, in your working directory. This should specify at least the `module` name, typically the name of your package. So you should have something like this,

```
module = "fontastic"
```

We now need to edit this file to make l3build aware of the Lua `fntbuild` scripts provided by fontscripts.

```
module = "fontastic"
require(kpse.lookup("fntbuild.lua"))
```

This should be sufficient to make the functions provided by the script available. To effectively use these functions, however, you will probably need to provide configuration details for your specific workflow and setup. For example, if you want to build fonts this way, you may need to specify which build tool you use and which files contain the code to compile or commands to execute. The remainder of this document explains how to do this[2]. You also need to tell the scripts if you want to override the defaults e.g. if you would like `fntbuild` to insert subset encoding declarations and/or variable scaling factors[3] into font definition files or to auto-generate tests and/or font tables.

[2] Unless you are reading a version which includes the implementation section. In that case, *most* of the document is just code listing and it is the first bit of what remains which explains how to actually use the scripts.

[3] `fontinst` only. As far as I know, this is simply an oversight on fontinst's part – fixed scaling factors

2 Lua Script Fragments

fntbuild.lua provides two custom l3build targets and five new Lua functions. In addition, it redefines three l3build functions (two of which do nothing by default). As well as those provided by l3build, a number of additional variables are used by these functions to determine what, where and how they should operate.

2.1 l3build Targets

`fnttarg` (*targ*) Runs `fnt.fontinst ()` by default or `fnt.afm2tfm ()` if `fnt.afmtotfm` is `true`.

Builds traditional TEX font files for 8-bit engines. The building is done in `fnt.fntdir`. The results are copied to `fnt.keepdir` and the intermediate results to `fnt.keeptempdir`. Files in the former are copied back into the `builddir` when building documentation, running tests etc.

This target may be redefined if another tool chain is used for font creation[4].

If additional setup is required, this may be done by defining `fnt.buildinit_hook ()`, which is empty by default. See section 2.2.1.

`uniquifyencs` (*targ*) ⟨*encoding tag*⟩

Tries to ensure that the names of encoding files and font encodings are unique by editing font definition files and map file fragments in `fnt.keepdir`. ⟨*encoding tag*⟩ is optional. If no argument is given and `encodingtag` is empty, a suitable tag will be determined automatically[5].

Note that it is not necessary to use this target if l3build fnttarg is used with a tool for which built-in support is provided.

2.2 Lua Functions

2.2.1 New Functions

`fnt.afm2tfm()` (*fn*) `fnt.afm2afm(`*<dir>*`)`

A function to generate support files for simple symbolic fonts in ⟨*dir*⟩ using `afm2tfm`. The function generates TFMs, FDs and a MAP. It requires AFMs and, optionally, ENCs. If ⟨*dir*⟩ is unspecified, `fnt.fntdir` is used.

If `fnt.afmtotfm` is `true`, a call to `fnt.afm2tfm()` replaces the default call to `fnt.fontinst()` when the `fnttarg` target is used.

Like `fnt.fontinst()`, `fnt.afm2tfm()` calls `build_fnt()` and `fnt.fntkeeper()`, but it does **not** call `fnt.uniquify()`.

Note this function is designed for only very simple cases. The target TEX fonts must each correspond to a unique .afm (though it would not be hard to extend the function to support a one-many mapping).

are supported out-of-the-box – and I assume other tools have the ability to include variable scaling without `fntbuild`'s interference.

[4] `fnt.afm2tfm ()` in fntbuild.lua and build.lua in berenisadf provide examples.

[5] Hopefully.

`fnt.buildinit()` (*fn*) `fnt.buildinit()`

Sets up the build environment in preparation for building by unpacking and copying files, installing dependencies etc. in preparation for building. Workflows supported out-of-the-box use this function, so there is no need to invoke it explicitly in these cases[6]. This function is intended for use in defining custom build functions. See section 5 for an example. It is called, if needed, by those provided out-of-the-box such as `fnt.fontinst()`.

If `fnt.needs_fontinst` is `true`, the files in `${TEXMFDIST}/tex/fontinst` are copied to `fnt.fntdir` before user-specified system files are processed. For files provided by fontscripts, the prefix `fontscripts-` is removed unless doing so conflicts with an existing file name.

Note that fontinst files may be used by tools other than fontinst. For example, otftotfm can use converted fontinst encoding files.

`fnt.buildinit_hook()` (*fn*) `fnt.buildinit_hook()`

A function which does nothing successfully by default, similar to the hooks provided by l3build. May be redefined to perform additional tasks when setting up the environment for `fnttarg`. The function should return an error level.

The hook is executed after unpacking, copying etc. at the end of `fnt.buildinit()` so it can be used to modify those preparations, if necessary.

`fnt.build_fnt()` (*fn*) `fnt.build_fnt(`*<dir>*`, `*<cmd>*`, `*<file>*`)`

A wrapper around `runcmd()` which runs ⟨*cmd*⟩ ⟨*file*⟩ in ⟨*dir*⟩. If `fnt.buildsearch` is `false` (the default), the function first sets up a restricted environment designed to avoid contaminating the build with undesirable elements from TEX search paths.

Note that the restrictions are designed to avoid contamination resulting from searches using `kpathsea`. If you use this function with a different tool chain, you may need to take additional steps to isolate the build. For example, the function does nothing to prevent a FontForge script accessing resources in `~/.local/share`, `/usr/local/fonts`, `/etc`, `/System/Library` etc.

In general, l3build 'sandboxing' uses a black-listing rather than a white-listing model. Since this is inherently less reliable, you should not depend on the effectiveness of isolation. This means you should check what actually happens e.g. which paths actually end up in your `.log` files or test the effectiveness of the isolation.

The table `fnt.build_fnt_env` may be used to selectively adapt the default sandboxing, which is sets the `kpathsea` variables shown in table 1.

`fnt.buildinit_tidy()` (*fn*) `fnt.buildinit_tidy()`

Removes `fnt.buildsuppfiles_sys` from `fnt.fntdir`. Note that this does not remove files installed by `fnt.buildinit_fontinst()`.

This function is intended for use in defining custom build functions. It is called, if needed, by those provided out-of-the-box such as `fnt.fontinst()`.

`fnt.finst()` (*fn*) `fnt.finst(`*<patt>*`, `*<dir>*`, `*<mode>*`)`

A wrapper around `fnt.build_fnt()` which runs `pdftex --interaction` ⟨*mode*⟩ in directory ⟨*dir*⟩ on each file in ⟨*dir*⟩ matching the pattern ⟨*patt*⟩.

[6] See `fnt.fontinst()` and `fnt.afm2tfm()` in fntbuild-build.lua.

Table 1: Default environment for 'sandboxed' build.

kpathsea variable	value
TEXINPUTS	. fnt.localtexmf()
TEXMFAUXTREES	{}
TEXMFHOME	{}
TEXMFLOCAL	{}
TEXMFCONFIG	.
TEXMFVAR	.
VFFONTS	${TEXINPUTS}
TFMFONTS	${TEXINPUTS}
TEXFONTMAPS	${TEXINPUTS}
T1FONTS	${TEXINPUTS}
AFMFONTS	${TEXINPUTS}
TTFFONTS	${TEXINPUTS}
OPENTYPEFONTS	${TEXINPUTS}
LIGFONTS	${TEXINPUTS}
ENCFONTS	${TEXINPUTS}

Called by `fnt.fontinst()`. There is usually no need to call this function directly.

`fnt.fontinst()` (*fn*) `fnt.fontinst(<dir>, <mode>)`

Executes a fontinst workflow to generate TEX fonts and associated files in directory ⟨*dir*⟩, executing `pdftex` with the option `--interaction` ⟨*mode*⟩.

This is the default build function called by `fnttarg`.

The function calls `fnt.finst()`, `fnt.fntkeeper()` and `fnt.uniquify()` in addition to performing initial compilation. The process requires the usual fontinst setup i.e. a driver to generate the initial files and a second file to create map file fragments.

If a variable scaling factor is used, the font definition files will be edited to ensure this works, since fontinst apparently supports scaling only by a fixed factor. For a simple demonstration of how to set this up, see, for example, baskervaldadf's driver which includes

```
% \declaresize{}{<-> \string\ybv@@scale}
```

Shape declarations should then be written without scaling,

```
% \installfont{ybvr8t}{ybvr8r,ybvr8sr,newlatin}{t1-baskervald}{T1}{ybv}{m}{n}{}
```

since scaling will be added after the font definition files are created,

```
% \expandafter\ifx\csname ybv@scale\endcsname\relax
%   \let\ybv@@scale\@empty
% \else
%   \edef\ybv@@scale{s*[\csname ybv@scale\endcsname]}%
% \fi
%
% \DeclareFontFamily{T1}{ybv}{}
```

```
%
% \DeclareFontShape{T1}{ybv}{m}{n}{
%   <-> \ybv@@scale ybvr8t
% }{}
```

to enable a suitable option in the `sty` file.

If fontinst is setup to produce a shell script for converting .pl and .vpl files to .tfm and .vf, the function executes the commands contained in this script. In order to maintain compatibility on Windows, the commands are extracted and run by Lua rather than calling the shell to execute the commands directly.

`fnt.fntsubsetter ()` (*fn*) `fnt.fntsubsetter()`

Inserts a font encoding subset declarations into font definition files. Explicit use is only required for custom build functions, as the standard options for `fnt.build_fnt ()` incorporate this function. Returns an error level, but this may not be accurate.

Note that, by default, the presence of this function does nothing because subset declarations are disabled by default. To insert declarations, not only must `fnt.fntsubsetter ()` be executed by the build function. `fnt.subset` must also be set to a non-`nil` value other than `false`.

`fnt.fntkeeper()` (*fn*) `fnt.fntkeeper(`*<dir>*`)`

Copies generated files in ⟨*dir*⟩ to `fnt.keepdir` and/or `fnt.keeptempdir` to prevent deletion by `l3build`. Returns an error level.

This function is called by the function executed by the default functions called by `fnttarg()` and may be useful if that target is redefined to call a different function.

`fnt.lsrdir()` (*fn*) `fnt.lsrdir(`*<path>, <filenames>*`)`

Adds a recursive list of files in `path` to the list of files in `filenames` and returns `filenames` and returns the resulting table. `path` should be a string containing the fully-qualified path to a directory. If specified, `filenames` should be a table. Note that the function appends entries to `filenames` rather than replacing them.

`fnt.test()` (*fn*) `fnt.test(`*<fntpkgname>, <fds>, <content>, <maps>, <fdsdir>*`)`

Auto-generates `lvt` files suitable for use with `l3build check` from a template.

`fnt.uniquify()` (*fn*) `fnt.uniquify(`*<tag>*`)`

Tries to ensure the names of font encodings and encoding files are unique by editing font definition files and map file fragments.

This function is used by `fnt.fontinst()` and `fnt.afm2tfm ()`, may be utilised in a custom definition of `fnt.build_fnt ()` or may be called directly using the `uniquifyencs` target.

2.2.2 Redefined Functions

`checkinit_hook()` (*fn*) `checkinit_hook()`

This is a standard l3build function which does nothing by default. fntbuild.lua redefines it to automatically generate test files suitable for use with `l3build check` if a test template is available. If `checksearch` is `false`, the hook also sets up an environment tailored to

Table 2: Default additions to 'sandbox' if none are specified. For files, the file is added. For directories, the contents are added.

Files		Directories
/tex/latex/base/	article.cls	/tex/latex/l3build
	fontenc.sty	/tex/latex/l3backend
	omlcmm.fd	/fonts/enc/dvips/base
	omlcmr.fd	/fonts/enc/dvips/cm-super
	omscmr.fd	/fonts/type1/public/amsfonts/cm
	omscmsy.fd	/fonts/type1/public/cm-super
	ot1cmr.fd	/fonts/tfm/public/cm
	ot1cmss.fd	/fonts/tfm/jknappen/ec
	ot1cmtt.fd	
	size10.clo	
	size11.clo	
	size12.clo	
	t1cmr.fd	
	t1cmss.fd	
	t1cmtt.fd	
	tracefnt.sty	
	ts1cmr.fd	
	ts1cmss.fd	
	ts1cmtt.fd	
	ucmr.fd	
	ucmss.fd	
	ucmtt.fd	
/tex/latex/fonttable/	fonttable.sty	

font-testing.

Note that `checksearch` is disabled by default. Contrary to l3build, `fntbuild.lua` enables sandboxing during both building and testing. If you do not want this, reenable `checksearch` after fntbuild.lua is read or, better yet, create a custom fntbuild-config.lua.

The default environment for testing does the following:

- concatenates all files specified in `fnt.mapfiles_sys` and writes the result to pdftex .map in `testdir`;
- copies the contents of `fnt.keepdir` to `testdir`;
- copies either the files specified in `fnt.checksuppfiles_sys` or the contents of the directories listed in table 2 to `testdir`;
- copies any files specified in `fnt.checksuppfiles_add` to `testdir`;
- copies the file specified by `fnt.regress` to `testdir`;
- copies the template specified by `fnt.testtemp` to `unpackdir` for use in auto-generating tests;
- restricts the environment by setting the variables listed in table 3.

Table 3: Default environment for 'sandboxed' testing.Note this configuration is applied *in addition* to the steps taken by `l3build -check` to 'isolate' the test environment when `checksearch` is `false`.

kpathsea variable	value
TEXMFAUXTREES	{}
TEXMFHOME	{}
TEXMFLOCAL	{}
TEXMFCONFIG	.
TEXMFVAR	.
VFFONTS	. fnt.localtexmf()
TFMFONTS	. fnt.localtexmf()
TEXFONTMAPS	. fnt.localtexmf()
T1FONTS	. fnt.localtexmf()
AFMFONTS	. fnt.localtexmf()
TTFFONTS	. fnt.localtexmf()
OPENTYPEFONTS	. fnt.localtexmf()
LIGFONTS	. fnt.localtexmf()
ENCFONTS	. fnt.localtexmf()

`copyctan()` (*fn*) `copyctan()`

This is extended to copy files from `fnt.keepdir` and to impose a single-layer of sub-directories of the kind required by CTAN for font distributions.

`docinit_hook()` (*fn*) `docinit_hook()`

This is a standard l3build function which does nothing by default. fntbuild.lua redefines it to automatically generate font tables suitable for use with `l3build doc` from a template.

2.3 Variables

Defaults assigned by fntbuild.lua to both l3build and fontscripts variables are listed in table 4 and table 5 respectively.

`fnt.afmtotfm` (*var*) `fnt.afmtotfm=` ⟨*true*/*false*⟩

`nil` by default. Setting this to anything other than `false` or `nil` will use `afm2tfm` rather than fontinst to build the fonts. This is only designed for ***extremely*** simple fonts such as those which contain only text symbols.

`fnt.autotestfds` (*var*) `fnt.autotestfds = {`*<globs>*`}`

`fnt.binmakers` (*var*) `fnt.binmakers = {`*<globs>*`}`

Scripts to run to convert human-readable TEX font metrics/virtual font metrics into binary TEX font metrics and virtual fonts.

`fnt.buildsearch` (*var*) `fnt.buildsearch=` ⟨*true*/*false*⟩

Whether to search the `texmf` trees while generating font files.

Defaults to `false`.

Table 4: fontscripts defaults for l3build variables.

Variable		Value
binaryfiles	=	{"*.pdf", "*.zip", "*.vf", "*.tfm", "*.pfb", "*.pfm", "*.ttf", "*.otf", "*.tar.gz"}
checkdeps	=	{maindir .. "/fnt-tests"}
checkengines	=	{"pdftex"}
checkformat	=	"latex"
checksearch	=	false
cleanfiles	=	{fnt.keeptempfiles}
installfiles	=	{"*.afm", "*.cls", "*.enc", "*.fd", "*.map", "*.otf", "*.pfb", "*.pfm", "*.sty", "*.tfm", "*.ttf", "*.vf"}
sourcefiledir	=	sourcefiledir or "."
sourcefiles	=	{"*.afm", "afm/*.afm", "*.pfb", "*.pfm", "*.dtx", "*.ins", "opentype/*.otf", "*.otf", "tfm/*.tfm", "truetype/*.ttf", "*.ttf", "type1/*.pfb", "type1/*.pfm"}
tdslocations	=	{ "fonts/afm/" .. fnt.vendor .. "/" .. module .. "/" .. "*.afm", "fonts/enc/dvips/" .. module .. "/" .. "*.enc", "fonts/map/dvips/" .. module .. "/" .. "*.map", "fonts/opentype/" .. fnt.vendor .. "/" .. module .. "/" .. "*.otf", "fonts/tfm/" .. fnt.vendor .. "/" .. module .. "/" .. "*.tfm", "fonts/truetype/" .. fnt.vendor .. "/" .. module .. "/" .. "*.ttf", "fonts/type1/" .. fnt.vendor .. "/" .. module .. "/" .. "*.pfb", "fonts/type1/" .. fnt.vendor .. "/" .. module .. "/" .. "*.pfm", "fonts/vf/" .. fnt.vendor .. "/" .. module .. "/" .. "*.vf", "source/fonts/" .. module .. "/" .. "*.etx", "source/fonts/" .. module .. "/" .. "*.mtx", "source/fonts/" .. module .. "/" .. "*-drv.tex", "source/fonts/" .. module .. "/" .. "*-map.tex", "tex/latex/" .. module .. "/" .. "*.fd", "tex/latex/" .. module .. "/" .. "*.sty" }
typesetexe	=	"TEXMFDOTDIR=.:../local: pdflatex"
typesetfiles	=	typesetfiles or {"*.dtx", "*-tables.tex", "*-example.tex"}
typesetsourcefiles	=	{fnt.keepdir .. "/*"}

Table 5: Default values for fontscripts variables.

Variable		Value
fnt.afmtotfm	=	nil
fnt.autotestfds	=	fnt.autotestfds or {}
fnt.binmakers	=	fnt.binmakers or {"*-pltotf.sh"}
fnt.builddeps	=	fnt.builddeps or {}
fnt.buildfiles	=	fnt.buildfiles or { "*.afm", "*.enc", "*.etx", "*.fd", "*.lig", "*.make", "*.map", "*.mtx", "*.nam", "*.otf", "*.pe", "*.tex" , "*.tfm" }
fnt.buildsearch	=	false
fnt.buildsuppfiles_sys	=	fnt.buildsuppfiles_sys or {}
fnt.checksuppfiles_sys	=	fnt.checksuppfiles_sys or {}
fnt.checksuppfiles_add	=	fnt.checksuppfiles_add or {}
fnt.familymakers	=	fnt.familymakers or {"*-drv.tex"}
fnt.fntdir	=	fnt.fntdir or builddir .. "/fnt"
fnt.fnttestfds	=	fnt.fnttestfds or {}
fnt.keepdir	=	fnt.keepdir or sourcefiledir .. "/keep"
fnt.keepfiles	=	fnt.keepfiles or {"*.enc", "*.fd", "*.map", "*.tfm", "*.vf"}
fnt.keeptempdir	=	fnt.keeptempdir or sourcefiledir .. "/keeptemp"
fnt.keeptempfiles	=	fnt.keeptempfiles or {"*.mtx", "*.pl", "*-pltotf.sh", "*-rec.tex", "*.vpl", "*.zz"}
fnt.mapmakers	=	fnt.mapmakers or {"*-map.tex"}
fnt.mapfiles_sys	=	fnt.mapfiles_sys or {}
fnt.mapfiles_add	=	fnt.mapfiles_add or {}
fnt.needs_fontinst	=	true
fnt.pkgbase	=	fnt.pkgbase or ""
fnt.regress	=	fnt.regress or "fntbuild-regression-test.tex"
fnt.subset	=	false
fnt.subsetdefns	=	fnt.subsetdefns or {}
fnt.subsetfiles	=	fnt.subsetfiles or {}
fnt.subsettemplate	=	fnt.subsettemplate or "\DeclareEncodingSubset{TS1}{$FONTFAMILY}{$SUBSET}"
fnt.tablestemp	=	fnt.tablestemp or "fntbuild-tables.tex"
fnt.testtemp	=	fnt.testtemp or "fntbuild-test.lvt"
fnt.vendor	=	fnt.vendor or "public"

Note that this is akin to l3build's `checksearch` etc. but for the additional build step required when preparing fonts.

`fnt.builddeps` (*var*) `fnt.builddeps = {}`

Dependencies to install when building fonts in a sandbox. If empty, some basic dependencies are installed to ensure straightforward cases work out-of-the-box.

`fnt.buildfiles` (*var*) `fnt.buildfiles = {`*<globs>*`}`

Source files to copy to the build directory when generating font files.

`fnt.buildsuppfiles_sys` (*var*) `fnt.buildsuppfiles_sys = {`*<globs>*`}`

`fnt.checksuppfiles_add` (*var*) `fnt.checksuppfiles_add = {`*<globs>*`}`

Dependencies to install when checking. Supplements default list.

`fnt.checksuppfiles_sys` (*var*) `fnt.checksuppfiles_sys = {`*<globs>*`}`

Dependencies to install when checking. Overrides default list.

`fnt.familymakers` (*var*) `fnt.familymakers = {`*<globs>*`}`

Source files `fnt.fontinst()` should compile to generate TEX support files.

For the default definition of `fnt.fontinst()` this variable should specify the driver or drivers to be compiled.

`fnt.fntdir` (*var*) `fnt.fntdir= "`⟨*directory path*⟩`"`

Directory in which to build fonts.

`fnt.fnttestfds` (*var*) `fnt.fnttestfds = {`*<globs>*`}`

Files to use when generating test files for `l3build`.

`fnt.keepdir` (*var*) `fnt.keepdir = "`⟨*dir*⟩`"`

Directory to store final products of font creation e.g. font definitions, map file fragments, TEX font metrics, virtual fonts etc.

`fnt.keepfiles` (*var*) `fnt.keepfiles = {`*<globs>*`}`

Files to copy to `fnt.keepdir`.

`fnt.keeptempdir` (*var*) `fnt.keeptempdir = "`⟨*dir*⟩`"`

Directory to store intermediat products of font creation e.g. human-readable TEX font metrics, virtual font metrics etc.

`fnt.keeptempfiles` (*var*) `fnt.keeptempfiles = {`*<globs>*`}`

Files to copy to `fnt.keeptempdir`.

`fnt.mapfiles_sys` (*var*) `fnt.mapfiles_sys = {`*<globs>*`}`

Map file fragments to install when checking. If unset, cm.map, cm-super-t1.map, cm-super-ts1.map and lm.map are used.

`fnt.mapfiles_add` (*var*) `fnt.mapfiles_add = {`*<globs>*`}`

Additional map file fragments to install when checking.

`fnt.mapmakers` (*var*) `fnt.mapmakers = {`*<globs>*`}`

Source files `fnt.finst()` should compile to generate map file fragments etc.

`fnt.needs_fontinst` (*var*) `fnt.needs_fontinst = ⟨true/false⟩`

`true` by default. Determines whether .etx and .mtx files under `tex/fontinst` in the distribution's TDS tree are copied to `fnt.fntdir` if `fnt.buildinit()` is called.

Unless you are using files with conflicting names (or whose names would conflict if the prefix `fontscripts-` was removed from those supplied by this package), it is probably best to change the default only if you are certain you do not need the files, you have set `fnt.buildsearch true` or you run into trouble. Note that not only `fontinst` make use of the files installed by fontinst and afm2pl.

`fnt.pkgbase` (*var*) `fnt.pkgbase = "⟨identifier⟩"`

May be used to set a string identifying the package if `fntbuild` has trouble determining one automatically[7].

`fnt.regress` (*var*) `fnt.regress= "⟨filename⟩"`

The file containing regression tests for use in auto-generating tests. fntbuild-regression-test.tex provides a set used by default.

Warning! The implementation of subset declarations in font definition files appears to be broken in current LaTeX[8]. Do NOT remove declarations from .sty files unless present in the format itself.

`fnt.subset` (*var*) `fnt.subset= ⟨true/false⟩`

`nil` by default. Set to any value other than `nil` or `false` to enable subset declarations.

Note this variable must be given a non-default value if subset encoding declarations should be added. Specifying definitions, files and templates only determines what will be done IF anything is done at all.

`fnt.subsetdefns` (*var*) `fnt.subsetdefns = {}`

If different families should get different subset values, the various settings should be configured here.

For example, the following code sets the subset for ⟨*family 1*⟩ to `1` and that for ⟨*family 2*⟩ to `3`.

```
fnt.subsetdefns.<family 1> = "1"
fnt.subsetdefns.<family 2> = "3"
```

`fnt.subsetfiles` (*var*) `fnt.subsetfiles = {<globs>}`

Font definition files into which encoding subset declarations should be inserted.

`fnt.subsettemplate` (*var*) `fnt.subsettemplate = "⟨⟩"`

Template to use when constructing subset declarations.

By default, the template simply plugs family and value pairs into a standard declaration as shown in table 5. However, it is also possible to use more complex declarations. For example, cfr-lm does not actually include any `TS1` fonts at all. Instead, it just says which

[7] None of the packages I've prepared with `fntbuild` have required this. Usually, the script should do a reasonable job of discovering a suitable value automatically.

[8] GitHub #1669.

Latin Modern family should be used for which cfr-lm family. So the only sensible approach is to use whichever subset declarations the relevant Latin Modern font definitions use. Unfortunately, these are provided by the LaTeX 2_ε format rather than lm, besides which it is at least theoretically possible the appropriate subsets might change and cfr-lm has no control over this. For this reason, the package simply says 'for cfr-lm family x, use whichever subset is declared for Latin Modern family y' for each pair of families x and y.

This is setup using the following line in the build.lua:

```
fnt.subsettemplate = "\\ExpandArgs {nnc} \\DeclareEncodingSubset {TS1} {$FONTFAMILY} {
```

Rather than numerical values, therefore, cfr-lm declares subset encodings such as

```
fnt.subsetdefns.clms = "lmss"
```

The result is that LaTeX 2_ε will use whichever subset is declared for `lmss` for `clms`, which, after all, is no more than a fancy alias for `lmss` in the `TS1` encoding.

`fnt.tablestemp` (*var*) `fnt.tablestemp= "⟨filename⟩"`

The file containing the template to use when auto-generating font tables. fntbuild-tables.tex provides the default template.

`fnt.testtemp` (*var*) `fnt.testtemp= "⟨filename⟩"`

The file containing the template to use when auto-generating font tests. fntbuild-test.lvt provides the default template.

`fnt.vendor` (*var*) `fnt.vendor = "⟨fnt.vendor⟩"`

Vendor directory for font installation such as `public` or `arkandis`.

3 Configuration

For simple cases, configuration may be done in build.lua, along with generic l3build customisation. For more elaborate modifications — or cases where a subset of configuration options should be shared between modules — settings may be placed in one or more fntbuild-config.lua files.

1. fntbuild-vars.lua is loaded. This ensures required variables get default values and modifies a number of l3build defaults.
2. A `kpse`-based search is performed. If fntbuild-config.lua is found, it is loaded.
3. If it exists, `maindir/fntbuild-config.lua` is loaded.
4. If it exists, `sourcedir/fntbuild-config.lua` is loaded.

So more specific settings will override more general ones.

4 Templates

By default, fntbuild.lua is able to utilise two kinds of `tex` template. The names of these files are stored in `fnt.testtemp` and `fnt.tablestemp`. If no local alternatives are provided, `fntbuild.lua` will use kpathsea to locate them, falling back to those included in the package and installed in `tex/latex/fontscripts/` under the `TEXMFDIST` tree by default.

The default files may be overridden by either or both of the following methods:

1. altering the values of `fnt.testtemp` and/or `fnt.tablestemp`;
2. providing local copies of the files named in `fnt.testtemp` and/or `fnt.tablestemp`.

The default contents are listed in section 5.2.

4.1 Font Tables

A template for producing font tables as part of package documentation is provided. The template is used in fntbuild.lua's `doc_init()` hook to generate `tex` files populated with font information from font definition files. This is then compiled by `l3build doc` to produce font tables.

Alternative content may be provided by supplying a local copy of fntbuild-tables.tex or specifying an alternative template in `fnt.tablestemp`. If `TABLES` occurs in the template, fntbuild.lua will replace it with a `document` environment containing a series of commands of the form `\sampletable{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`, where ⟨*`encoding`*⟩, ⟨*`family`*⟩, ⟨*`series`*⟩ and ⟨*`shape`*⟩ are derived from the font definition files.

The default template is listed in section 5.2.1.

4.2 Font Tests

If a template is found in the font test directory, it will be used in fntbuild.lua's `check_init()` hook to generate `lvt` files populated with font information from font definition files. These tests are then compiled by `l3build check` as part of the test suite. Certain file patterns are excluded from testing. In particular, separate tests are not generated for `ts1 fd` files because these families are typically better tested along with their `t1` counterparts.

Alternative content may be provided by supplying a local copy of fntbuild-test.lvt or specifying an alternative template in `fnt.testtemp`. If `SAMP` occurs in the template, fntbuild.lua will replace it with a series of tests derived from the font definition files. These tests should typically use commands provided explicitly or implicitly by the file specified in `fnt.regress`. By default, tests utilise a macro `\sampler` defined to absorb four arguments of the form `{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`. The result is a document containing various pieces supplied by fonttable.

A custom test suite may be specified by supplying a local copy of fntbuild-regression-test.tex, giving an alternative in `fnt.regress` or defining a different set of tests in the template specified by `fnt.testtemp`.

Note that you MUST supply a custom template if you change `fnt.regress`. It is, however, fine to supply a customised fntbuild-regression-test.tex for use with the default template. Only if you want to use a different filename for the test suite is a custom template required.

The default template is listed in section 5.2.2. The default test suite is listed in section 5.3.

5 Customisation

As in the case of l3build, you can replace functions and targets at will, albeit on a much more limited scale. Although it would be better to just not use fntbuild.lua at all if you want to redefine everything, it can make sense to replace `fnt.fontinst()` if, say, you want to use different font creation tools but make use of the functions for stashing generated files, generating font tables, testing etc.

For example, berenisadf was built using a build.lua containing

```
-- require(kpse.lookup("fntbuild.lua"))
local function fntmake (dir,mode)
  dir = dir or fnt.fntdir
  mode = mode or "errorstopmode --halt-on-error"
  local autotcfds ={ "ts1ybd2j.fd", "ts1ybd2.fd", "ts1ybdj.fd", "ts1ybd.fd" }
  -- set up the build environment
  assert(fnt.buildinit(), "Setting up build environment failed!")
  print("Running make. Please be patient ...\n")
  assert(run(dir, "chmod +x ff-ybd.pe"),"Attempt to make fontforge script executable i
  assert(run(dir, "make -f Makefile.make all"), '"make -f Makefile.make all" failed in
  -- make ts1 swash families so tc commands pick up the characters in ly1
  -- we don't need t1 versions of these families as there's no room for swash
  -- there
  -- ideally, we could just tell latex to use the non-swash families for the
  -- tc encoding, but that doesn't seem possible without rewriting more
  -- internal stuff than seems altogether wise ...
  for i, j in ipairs(autotcfds) do
    local jfam = string.gsub(j, "^ts1(.*)%.fd$", "%1")
    local jnewfam = jfam .. "w"
    local jnew = string.gsub(j, "(%.fd)$", "w%1")
    local f = assert(io.open(dir .. "/" .. j,"rb"))
    local content = f:read("*all")
    f:close()
    -- normalise line endings to be platform-agnostic
    -- copied from l3build
    content = string.gsub(content .. (string.match(content,"\n$") and "" or "\n"),
    "\r\n", "\n")
    local new_content = string.gsub(content,
    "{" .. jfam .. "}", "{" .. jnewfam .. "}")
    new_content = string.gsub(new_content, "(ts1[^%.]*)(%.fd)", "%1w%2")
    new_content = string.gsub(new_content, "(TS1/ybd[a-z0-9]*)", "%1w")
    f = assert(io.open(dir .. "/" .. jnew,"w"))
    f:write((string.gsub(new_content,"\n",fnt.os_newline_cp)))
```

```
    f:close()
  end
  -- call fnt.fntkeeper() to save the build results into fnt.keepdir else
  -- l3build deletes them before testing or compilation!
  assert(fnt.fntkeeper(),"FONT KEEPER FAILED IN " .. dir .. "! DO NOT MAKE STANDARD TA
  return 0
end
-- make local function available in table bt
bt = {}
bt.fntmake = fntmake
-- redefine fnt.ntarg so that fnttarg calls bt.fntmake rather than fontinst
-- fntmake must be specified first
-- it just ain't TeX
target_list[fnt.ntarg] = {
  func = bt.fntmake,
  desc = "Creates TeX font files",
  pre = function(names)
    if names then
      print("fntmake does not need names\n")
      help()
      exit(1)
    end
    return 0
  end
}
```

That is, berenisadf doesn't use any built-in build function at all, but `fnt.fntmake()`, which simply invokes `gnu make` and calls `fnt.fntkeeper()`[9]. Note the use of `fnt.buildinit()` to setup the build environment.

5.1 Examples

The latest versions of the following packages were developed using fontscripts and build using l3build[10]. Full details, including the use of templates and scaling are available on CODEBERG or GITHUB as part of the code repository. ebgaramond-maths provides a further example.

- GUST:
  - cfr-lm
- ARKANDIS:
  - baskervaldadf
  - berenisadf
  - electrumadf
  - librisadf

[9] It does not ensure encoding names are unique because it uses no custom encodings of its own.
[10] Note that it is in no sense a dependency; it contains nothing ever required in typesetting.

- romandeadf
- venturisadf.

5.2 Templates

Templates for auto-generating font tables and regression tests.

5.2.1 Font Tables Template

`fntbuild-tables.tex` (*tpt*)

```
\pdftracingfonts=1
\RequirePackage{svn-prov}
\ProvidesFileSVN[fnt-tables.tex]{$Id: fontscripts.dtx 10843 2025-02-22 05:01:11Z
cfrees $}[v0.2 \revinfo][\filebase: font table template]
\DefineFileInfoSVN

\documentclass[10pt,a4paper]{article}
\usepackage{geometry}
\usepackage{fonttable}
\newcommand\sampletable[4]{%
  #1/#2/#3/#4:\par\noindent
  \xfonttable{#1}{#2}{#3}{#4}%
  \clearpage
}

TABLES

```

5.2.2 Test Template

`fntbuild-test.lvt` (*tpt*)

```
\RequirePackage{svn-prov}
\ProvidesFileSVN[fnt-test.lvt]{$Id: fontscripts.dtx 10843 2025-02-22 05:01:11Z cfrees
$}[v0.0 \revinfo][fontscripts: l3build test template]
\listfiles
\input regression-test.tex\relax
\input fntbuild-regression-test.tex\relax
\setlength{\parindent}{0pt}
\textheight=250mm
\textwidth=160mm
\oddsidemargin=0mm
\evensidemargin=0mm
\headheight=0pt
\headsep=0pt
\topmargin=-10mm
\marginparwidth=0pt
\marginparsep=0pt

SAMP

```

5.3 Regression Tests

Simple regression test suite for fonts focused on T1 and TS1 encodings.

build-regression-test.tex (*tpt*)

```
\pdftracingfonts=1
\RequirePackage{svn-prov}
\ProvidesFileSVN[fnt-tests.tex]{$Id: fontscripts.dtx 10843 2025-02-22 05:01:11Z
cfrees $}[v0.0 \revinfo][fontscripts: l3build tests]
\documentclass[10pt,a4paper]{article}
\usepackage{fonttable}
\usepackage{tracefnt}% infoshow is default; debugshow traces maths fonts, too
\PassOptionsToPackage{nfssext-cfr}{debug}
\newcounter{tccharcnt}
\NewDocumentCommand \tcfnttest { o m }
{%
  \iftctest
    \stepcounter{tccharcnt}{\normalfont\thetccharcnt:~}%
    % \texttt{\textbackslash #2}:
    \IfValueTF{#1}{%
      \csname #2\endcsname {#1}%
    }{%
      \csname #2\endcsname
    }%
  \fi
}
\newif\iftctest
\tctesttrue
\makeatletter
\newcommand\sampler{}
\def\sampler#1#2#3#4{%
  \setcounter{tccharcnt}{0}%
  % \tracingoutput=1
  #1 / #2 / #3 / #4 :

  \tracinglostchars\thr@@
  % \tracingonline\thr@@
  \showoutput
  \pikfont{#1}{#2}{#3}{#4}%
  \aztext

  \AZtext

  \digitstext

  \punctext

  \knutext

  Ææ Ŵŵ Ŷŷ Šš €

  percent: \%

```

```
\begin{tabular}{lllllllll}
  \tcfnttest{S} &
  \tcfnttest[I]{capitalacute} &
  \tcfnttest[I]{capitalbreve} &
  \tcfnttest[I]{capitalcaron} &
  \tcfnttest[I]{capitalcedilla} &
  \tcfnttest[I]{capitalcircumflex} &
  \tcfnttest[I]{capitaldieresis} &
  \tcfnttest[I]{capitaldotaccent} &
  \tcfnttest[I]{capitalgrave} &
  \tcfnttest[I]{capitalhungarumlaut} \\
  \tcfnttest[I]{capitalnewtie} &
  \tcfnttest[I]{capitalogonek} &
  \tcfnttest[I]{capitalring} &
  \tcfnttest[I]{capitaltie} &
  \tcfnttest[I]{capitaltilde} &
  \tcfnttest{textacutedbl} &
  \tcfnttest{textasciiacute} &
  \tcfnttest{textasciibreve} &
  \tcfnttest{textasciicaron} &
  \tcfnttest{textasciicircum} \\
  \tcfnttest{textasciidieresis} &
  \tcfnttest{textasciigrave} &
  \tcfnttest{textasciimacron} &
  \tcfnttest{textasciitilde} &
  \tcfnttest{textasteriskcentered} &
  \tcfnttest{textbackslash} &
  \tcfnttest{textbaht} &
  \tcfnttest{textbar} &
  \tcfnttest{textbardbl} &
  \tcfnttest{textbigcircle} \\
  \tcfnttest{textblank} &
  \tcfnttest{textborn} &
  \tcfnttest{textbraceleft} &
  \tcfnttest{textbraceright} &
  \tcfnttest{textbrokenbar} &
  \tcfnttest{textbullet} &
  \tcfnttest{textcelsius} &
  \tcfnttest{textcent} &
  \tcfnttest{textcentoldstyle} &
  \tcfnttest{textcircledP} \\
  \tcfnttest{textcolonmonetary} &
  \tcfnttest{textcompwordmark} &
  \tcfnttest{textcopyleft} &
  \tcfnttest{textcopyright} &
  \tcfnttest{textcurrency} &
  \tcfnttest{textdagger} &
  \tcfnttest{textdaggerdbl} &
  \tcfnttest{textdblhyphen} &
  \tcfnttest{textdblhyphenchar} &
  \tcfnttest{textdegree} \\
  \tcfnttest{textdied} &
  \tcfnttest{textdiscount} &
  \tcfnttest{textdiv} &
```

```
135     \tcfnttest{textdivorced} &
136     \tcfnttest{textdollar} &
137     \tcfnttest{textdollaroldstyle} &
138     \tcfnttest{textdong} &
139     \tcfnttest{textdownarrow} &
140     \tcfnttest{texteightoldstyle} &
141     \tcfnttest{textestimated} \\
142     \tcfnttest{texteuro} &
143     \tcfnttest{textfiveoldstyle} &
144     \tcfnttest{textflorin} &
145     \tcfnttest{textfouroldstyle} &
146     \tcfnttest{textfractionsolidus} &
147     \tcfnttest{textgravedbl} &
148     \tcfnttest{textguarani} &
149     \tcfnttest{textinterrobang} &
150     \tcfnttest{textinterrobangdown} &
151     \tcfnttest{textlangle} \\
152     \tcfnttest{textlbrackdbl} &
153     \tcfnttest{textleaf} &
154     \tcfnttest{textleftarrow} &
155     \tcfnttest{textlira} &
156     \tcfnttest{textlnot} &
157     \tcfnttest{textlquill} &
158     \tcfnttest{textmarried} &
159     \tcfnttest{textmho} &
160     \tcfnttest{textminus} &
161     \tcfnttest{textmu} \\
162     \tcfnttest{textmusicalnote} &
163     \tcfnttest{textnaira} &
164     \tcfnttest{textnaira} &
165     \tcfnttest{textnineoldstyle} &
166     \tcfnttest{textnumero} &
167     \tcfnttest{textohm} &
168     \tcfnttest{textonehalf} &
169     \tcfnttest{textoneoldstyle} &
170     \tcfnttest{textonequarter} &
171     \tcfnttest{textonesuperior} \\
172     \tcfnttest{textopenbullet} &
173     \tcfnttest{textordfeminine} &
174     \tcfnttest{textordmasculine} &
175     \tcfnttest{textparagraph} &
176     \tcfnttest{textperiodcentered} &
177     \tcfnttest{textpertenthousand} &
178     \tcfnttest{textperthousand} &
179     \tcfnttest{textpeso} &
180     \tcfnttest{textpilcrow} &
181     \tcfnttest{textpm} \\
182     \tcfnttest{textquotedblleft} &
183     \tcfnttest{textquotedblright} &
184     \tcfnttest{textquoteleft} &
185     \tcfnttest{textquoteright} &
186     \tcfnttest{textquotesingle} &
187     \tcfnttest{textquotestraightbase} &
188     \tcfnttest{textquotestraightdblbase} &
```

```
    \tcfnttest{textrangle} &
    \tcfnttest{textrbrackdbl} &
    \tcfnttest{textrecipe} \\
    \tcfnttest{textreferencemark} &
    \tcfnttest{textregistered} &
    \tcfnttest{textrightarrow} &
    \tcfnttest{textrquill} &
    \tcfnttest{textsection} &
    \tcfnttest{textservicemark} &
    \tcfnttest{textsevenoldstyle} &
    \tcfnttest{textsixoldstyle} &
    \tcfnttest{textsterling} &
    \tcfnttest{textsurd} \\
    \tcfnttest{textthreeoldstyle} &
    \tcfnttest{textthreequarters} &
    \tcfnttest{textthreequartersemdash} &
    \tcfnttest{textthreesuperior} &
    \tcfnttest{texttildelow} &
    \tcfnttest{texttimes} &
    \tcfnttest{texttrademark} &
    \tcfnttest{texttwelveudash} &
    \tcfnttest{texttwooldstyle} &
    \tcfnttest{texttwosuperior} \\
    \tcfnttest{textuparrow} &
    \tcfnttest{textwon} &
    \tcfnttest{textyen} &
    \tcfnttest{textzerooldstyle} &
                                &
                                &
                                &
                                &
                                &
                                \\
  \end{tabular}

  \vfil\break
}

\makeatother
```

Change History

0.3

General: Include warning re. use of subset declarations in .fd files. GitHub #1669. 12

v0.1

General: First public release. 1

v0.2

General: `fnttarg` may now be 'sandboxed'. The default build recipes (based on fontinst and `afm2tfm`) are configured to do this and `fnt.buildsearch` is disabled by default. At present, isolation only affects TEX-based build tools. FontForge and GNU `make` are unaffected, for example. 3

`fontinst.lua` renamed to `fntbuild.lua`. 3
Default templates. Simplified template configuration and search. 14
Disable `checksearch` by default. . . . 7
New function: `fnt.afm2tfm()`. 3
New function: `fnt.build_ fnt()`. . 4
New function: `fnt.buildinit()`. . . 3
New function: `fnt.buildinit_ hook()`. 4
New function: `fnt.fntsubsetter()`. 6
New function: `fnt.lsrdir()`. 6
New variable: `fnt.afmtotfm`. 8
New variable: `fnt.builddeps`. . . . 11
New variable: `fnt.buildfiles`. . . 11
New variable: `fnt.buildsearch`. . . 8
New variable: `fnt.buildsuppfiles_sys`. 11
New variable: `fnt.checksuppfiles_add`. 11
New variable: `fnt.checksuppfiles_sys`. 11
New variable: `fnt.fntdir`. 11
New variable: `fnt.mapfiles_add`. . . 11
New variable: `fnt.mapfiles_sys`. . . 11
New variable: `fnt.regress`. 12
New variable: `fnt.subsetdefns`. . . 12
New variable: `fnt.subsetfiles`. . . 12
New variable: `fnt.subsettemplate`. 12
New variable: `fnt.subset`. 12
New variable: `fnt.tablestemp`. . . 13
New variable: `fnt.testtemp`. . . . 13
Newly documented variable: `fnt.pkgbase`. 12
Renamed function: `fnt.finst()` (was `finst()`. 4
Renamed function: `fnt.fntkeeper()` (was `fntkeeper()`. 6
Renamed function: `fnt.fontinst()` (was `fontinst()`. 5
Renamed function: `fnt.test()` (was `fnt_test()`. 6
Renamed variable: `fnt.autotestfds` (was `autotestfds`). 8
Renamed variable: `fnt.binmakers` (was `binmakers`). 8
Renamed variable: `fnt.familymakers` (was `familymakers`). 11
Renamed variable: `fnt.fnttestfds` (was `fnttestfds`). 11
Renamed variable: `fnt.keepdir` (was `keepdir`). 11
Renamed variable: `fnt.keepfiles` (was `keepfiles`). 11
Renamed variable: `fnt.keeptempdir` (was `keeptempdir`). 11
Renamed variable: `fnt.keeptempfiles` (was `keeptempfiles`). 11
Renamed variable: `fnt.mapmakers` (was `mapmakers`). 11
Renamed variable: `fnt.vendor` (was `vendor`). 13
Support for configuration files. See section 3. 13
Use `fnt` prefix in Lua scripts. 1

v0.3

General: New function: `fnt.buildinit_ tidy()`. 4
New variable: `fnt.needs_fontinst`. 11

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

`\%` 79
`\\` 91, 101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 201, 211, 221

A

`\AZtext` 69
`\aztext` 67

B

`\begin` 81
`\break` 224

C

checkinit_hook() (fn) 6
\clearpage 11
copyctan() (fn) 8
\csname 48, 50

D

\def 58
\DefineFileInfoSVN 4
\digitstext 71
docinit_hook() (fn) 8
\documentclass 5, 37

E

\end 222
\endcsname 48, 50
\evensidemargin 25

F

\fi 52
filebase commands:
\filebase: 3
fnt.afm2tfm() (fn) 3
fnt.afmtotfm (var) 8
fnt.autotestfds (var) 8
fnt.binmakers (var) 8
fnt.build_fnt() (fn) 4
fnt.builddeps (var) 11
fnt.buildfiles (var) 11
fnt.buildinit() (fn) 4
fnt.buildinit_hook() (fn) 4
fnt.buildinit_tidy() (fn) 4
fnt.buildsearch (var) 8
fnt.buildsuppfiles_sys (var) 11
fnt.checksuppfiles_add (var) 11
fnt.checksuppfiles_sys (var) 11
fnt.familymakers (var) 11
fnt.finst() (fn) 4
fnt.fntdir (var) 11
fnt.fntkeeper() (fn) 6
fnt.fntsubsetter () (fn) 6
fnt.fnttestfds (var) 11
fnt.fontinst() (fn) 5
fnt.keepdir (var) 11
fnt.keepfiles (var) 11
fnt.keeptempdir (var) 11
fnt.keeptempfiles (var) 11
fnt.lsrdir() (fn) 6
fnt.mapfiles_add (var) 11
fnt.mapfiles_sys (var) 11
fnt.mapmakers (var) 11
fnt.needs_fontinst (var) 12
fnt.pkgbase (var) 12
fnt.regress (var) 12
fnt.subset (var) 12
fnt.subsetdefns (var) 12
fnt.subsetfiles (var) 12
fnt.subsettemplate (var) 12
fnt.tablestemp (var) 13
fnt.test() (fn) 6
fnt.testtemp (var) 13
fnt.uniquify() (fn) 6
fnt.vendor (var) 13
fntbuild-regression-test.tex (tpt) . . 34
fntbuild-tables.tex (tpt) 1
fntbuild-test.lvt (tpt) 16
fnttarg (targ) 3
functions:
checkinit_hook() 6
copyctan() 8
docinit_hook() 8
fnt.afm2tfm() 3
fnt.build_fnt() 4
fnt.buildinit() 4
fnt.buildinit_hook() 4
fnt.buildinit_tidy() 4
fnt.finst() 4
fnt.fntkeeper() 6
fnt.fntsubsetter () 6
fnt.fontinst() 5
fnt.lsrdir() 6
fnt.test() 6
fnt.uniquify() 6

H

\headheight 26
\headsep 27

I

\iftctest 44, 54
\IfValueTF 47
\input 19, 20

K

\knutext 75

L

\listfiles 18

M

\makeatletter 56
\makeatother 227
\marginparsep 30
\marginparwidth 29

N

\newcommand 8, 57
\newcounter 41
\NewDocumentCommand 42

\newif 54
\noindent 9
\normalfont 45

O

\oddsidemargin 24

P

\par 9
\parindent 21
\PassOptionsToPackage 40
\pdftracingfonts 1, 34
\pikfont 66
\ProvidesFileSVN 3, 17, 36
\punctext 73

R

\relax 19, 20
\RequirePackage 2, 16, 35
\revinfo 3, 17, 36

S

\sampler *14*, 57, 58
\sampletable *14*, 8
\setcounter 59
\setlength 21
\showoutput 65
\stepcounter 45

T

targets:
fnttarg *3*
uniquifyencs *3*
\tcfnttest 42, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215
\tctesttrue 55
templates:
fntbuild-regression-test.tex . . . 34
fntbuild-tables.tex 1
fntbuild-test.lvt 16
TEX and LATEX 2ε commands:
\thr@@ 63, 64
\textbackslash 46
\textheight 22
\texttt 46
\textwidth 23
thetccharcnt commands:
\thetccharcnt: 45
\topmargin 28
\tracinglostchars 63
\tracingonline 64
\tracingoutput 60

U

uniquifyencs (targ) *3*
\usepackage 6, 7, 38, 39

V

variables:
fnt.afmtotfm *8*
fnt.autotestfds *8*
fnt.binmakers *8*
fnt.builddeps *11*
fnt.buildfiles *11*
fnt.buildsearch *8*
fnt.buildsuppfiles_sys *11*
fnt.checksuppfiles_add *11*
fnt.checksuppfiles_sys *11*
fnt.familymakers *11*
fnt.fntdir *11*
fnt.fnttestfds *11*
fnt.keepdir *11*
fnt.keepfiles *11*
fnt.keeptempdir *11*
fnt.keeptempfiles *11*
fnt.mapfiles_add *11*
fnt.mapfiles_sys *11*
fnt.mapmakers *11*
fnt.needs_fontinst *12*
fnt.pkgbase *12*
fnt.regress *12*
fnt.subset *12*
fnt.subsetdefns *12*
fnt.subsetfiles *12*
fnt.subsettemplate *12*
fnt.tablestemp *13*
fnt.testtemp *13*
fnt.vendor *13*
\vfil 224

X

\xfonttable 10