The luaotfload package

LaTeX3 Project
Elie Roux · Khaled Hosny · Philipp Gesang · Ulrike Fischer · Marcel Krüger
Home: https://github.com/latex3/luaotfload

2022-10-03 v3.23

Abstract

This package is an adaptation of the ConTEXt font loading system. It allows for loading OpenType fonts with an extended syntax and adds support for a variety of font features.

After discussion of the font loading API, this manual gives an overview of the core components of Luaotfload: The packaged font loader code, the names database, configuration, and helper functions on the Lua end.

Contents

1 Engine and version support **3**

2 Changes **3**

2.1 New in version 3.23 . . . 3
2.2 New in version 3.22 . . . 3
2.3 New in version 3.21 (by Ulrike Fischer/Marcel Krüger) . . . 3
2.4 New in version 3.20 (by Ulrike Fischer/Marcel Krüger) . . . 3
2.5 New in version 3.19 (by Ulrike Fischer/Marcel Krüger) . . . 3
2.6 New in version 3.18 (by Ulrike Fischer/Marcel Krüger) . . . 4
2.7 New in version 3.17 (by Ulrike Fischer/Marcel Krüger) . . . 4
2.8 New in version 3.16 (by Ulrike Fischer/Marcel Krüger) . . . 4
2.9 New in version 3.15 (by Ulrike Fischer/Marcel Krüger) . . . 4
2.10 New in version 3.14 (by Ulrike Fischer/Marcel Krüger) . . . 5
2.11 New in version 3.13 (by Ulrike Fischer/Marcel Krüger) . . . 5
2.12 New in version 3.12 (by Ulrike Fischer/Marcel Krüger) . . . 5
2.13 New in version 3.11 (by Ulrike Fischer/Marcel Krüger) . . . 6
2.14 New in version 3.10 (by Ulrike Fischer/Marcel Krüger) . . . 6
2.15 New in version 3.00 (by Ulrike Fischer/Marcel Krüger) . . . 6
2.16 New in version 2.99 (by Ulrike Fischer) . . . 6
2.17 New in version 2.98 (by Ulrike Fischer) . . . 6
2.18 New in version 2.97 (by Ulrike Fischer) . . . 6
2.19 New in version 2.96 (by Ulrike Fischer) . . . 7
2.20 New in version 2.95 (by Ulrike Fischer) . . . 7
2.21 New in version 2.94 (by Ulrike Fischer) . . . 7
2.22 New in version 2.93 (by Ulrike Fischer) . . . 7
2.23 New in version 2.92 (by Ulrike Fischer) . . . 7
2.24 New in version 2.91 (by Ulrike Fischer) . . . 7

2.25 New in version 2.9 (by Ulrike Fischer) . . . 8
3 Introduction 8
4 Thanks 9
5 Loading Fonts 9
5.1 Prefix – the luaotfloadWay . . . 9
5.2 Bracketed Lookups . . . 10
5.3 Compatibility . . . 10
5.4 Examples . . . 11
5.4.1 Loading by File Name . . . 11
5.4.2 Loading by Font Name . . . 11
5.4.3 Modifiers . . . 12
6 Font features 12
6.1 Basic font features . . . 13
6.2 Non-standard font features . . . 21
7 Combining fonts 22
7.1 Fallbacks . . . 22
7.2 Combinations . . . 22
8 Font names database 23
8.1 luaotfload-tool . . . 23
8.2 Search Paths . . . 24
8.3 Querying from Outside . . . 24
8.4 Blacklisting Fonts . . . 25
9 The Fontloader 25
9.1 Overview . . . 25
9.2 Contents and Dependencies . . . 25
9.3 Packaging . . . 28
10 Configuration Files 29
11 Auxiliary Functions 29
11.1 Callback Functions . . . 30
11.1.1 Compatibility with Earlier Versions . . . 30
11.1.2 Patches . . . 30
11.2 Package Author's Interface . . . 31
11.2.1 Font Properties . . . 31
11.2.2 Database . . . 32
11.3 Format Author's Interface . . . 32
11.3.1 Color setting . . . 32
11.3.2 Color selection . . . 33
12 Troubleshooting 33
12.1 Database Generation . . . 33
12.2 Font Features . . . 33
12.3 LuaTEX Programming . . . 34
13 License 34
Appendix: Manual of luaotfload.conf 37
Appendix: Manual of luaotfload-tool 44

1 Engine and version support

luaotfload is a quite large and complex package. It imports code from context which is actively developed along with the luatex binary. It is not possible to support a large number of engines variants or versions.

Supported is the luatex versions of a current TeXLive 2019 (and a current MiKTeX). Beginning with version 3.1 of this package also luahbtex is supported.

2 Changes

2.1 *New in version 3.23*

- More reliable ToUnicode mappings in Harf mode
- Various bug fixes

2.2 *New in version 3.22*

- Tailored case mappings
- Avoid database rebuilds when switching Lua versions
- Improved attribute settings of ActualText nodes in harf mode
- Callback-based customization for color parameter
- Various bug fixes

2.3 *New in version 3.21 (by Ulrike Fischer/Marcel Krüger)*

- Fix performance regression introduced in version 3.19.
- More reliably support TrueType based variable fonts in `harf` mode.

2.4 *New in version 3.20 (by Ulrike Fischer/Marcel Krüger)*

- A bug in luaotfload-tool has been corrected.
- The directory for the font name database has been corrected and no longer uses the development directory.

2.5 *New in version 3.19 (by Ulrike Fischer/Marcel Krüger)*

- When used with LuaTeX 1.15.0 or newer, variable fonts are supported when using the `harf` shaper too.
- A new algorithm for selecting fonts based on font family names allows to more reliably load fonts based on their family name.
- The compiled font database gets compressed to reduce disk space and improve performance on newer systems.
- Broken rendering of some TrueType based variable fonts has been fixed.

- Text automatically gets normalized to Unicode's NFC before shaping. This improves rendering for text written in decomposed forms for many fonts. This can be turned off by passing the `-normalize` font feature.
- A number of small bugfixes.

2.6 *New in version 3.18 (by Ulrike Fischer/Marcel Krüger)*

- Now variable fonts can be loaded with default values without specifying any explicit axis values.
- A number of small bugfixes.

2.7 *New in version 3.17 (by Ulrike Fischer/Marcel Krüger)*

- The experimental support for OpenType variable fonts has been extended to more reliably support modern fonts.
- A number of small bugfixes.

2.8 *New in version 3.16 (by Ulrike Fischer/Marcel Krüger)*

- The entry point is called luaotfload.lua instead of luaotfload-main.lua (but the old name is still provided for compatibility).
- `pre/post_shaping_filter` callbacks has been added.
- The number of LUA-files and submodules shown in the LOG-file has been reduced. But it is extended again by setting the environment variable `LUAOTFLOAD_TRACE_SUBMODULES=1`.
- The HarfBuzz based shaper will in some situations drop hyphenation points. This happens less frequently now since the new version uses first/second discretionaries (the mechanism described in the LuaTeX manual, section 5.6 for the of-f-ice example) to support a limited amount of nesting.
- When the `node` shaper is used, experimental support for OpenType variable fonts has been added. To use them, set the font feature `axis` to a comma separated list of axis names and values. (E.g. `axis=weight=600`) The supported axis names and value range depend on the font (see page 20).
- The font features `upper` and `lower` can be used to map the text of a font to upper or lowercase before displaying it. Currently this implements the untailored Unicode case mapping algorithm, but it is planned to add tailoring later (see page 20).
- A number of small bugfixes.

2.9 *New in version 3.15 (by Ulrike Fischer/Marcel Krüger)*

- The font database is updated more reliably if fonts get deleted.
- In multiple error cases, error messages are shown instead of silently generating bad output.
- Write glyph ids instead of internal identifiers to DVI files. This allows using OpenType fonts when working with dvilualatex. (This requires additional support from the DVI reader)

Change!

- The set of font features which are enabled by default has been changed to be more similar to HarfBuzz. Especially “Above-base mark Positioning” (abvm), “Below-base mark Positioning” (blwm), “Contextual Alternates” (calt), “Cursive Positioning” (curs), “Distances” (dist), and “Required Contextual Alternates” (rclt) are now enabled by default for all scripts.
- Added a **mathfontdimen** font feature which allows emulating fontdimen values from xetex or traditional TEX math fonts.
- Initial support for variable fonts in **node** mode.

2.10 *New in version 3.14 (by Ulrike Fischer/Marcel Krüger)*

- a bug in luaotfload-tool has been corrected (reported on the texlive list)
- the fontloader has been patched to resolve a problem with color fonts and save/restore pairs (issue #124)

2.11 *New in version 3.13 (by Ulrike Fischer/Marcel Krüger)*

Change!

- A problem with text fonts with minimal math table has been fixed (issue #148): In new luaotfload versions, math parameters will only be loaded for fonts with `script=math`. If you do want to set math parameters for fonts with other scripts, add `-nomathparam`. We strongly recommend against setting math parameters for text fonts because these would overwrite parameters from actual math fonts.
- A bug in harf-mode that could lead to missing chars and freezing was corrected (issue #141).
- A font size problem in harf-mode has been fixed (issue #147).
- An error if the main function was called twice has been fixed (issue #145).
- Allow .ttf fonts to be loaded with a map file with luahbtex (issue #142) (issue #143).
- Fonts installed for a single user on windows are now found (issue #138).
- A problem with wrong TEX-ligatures in harf mode has been fixed (issue #139).
- The debugging output has been changed (issue #131).
- A missing “capital sharp s” (U+1E9E) in a font is replaced by SS instead of giving a missing character message: ß or SS
- The color handling has been improved so that it is now compatible with the luacolor package.

2.12 *New in version 3.12 (by Ulrike Fischer/Marcel Krüger)*

- Corrected a number of small bugs in harf mode.

Experimental!

- Extension of the **color** key to allow coloring of specific output glyphs, see page 14

Experimental!

- A new **fallback** key to allow to define fallback fonts, see page 19

Experimental!

- A new **multiscript** key to allow to use a font for more than one script, see page 17

2.13 *New in version 3.11 (by Ulrike Fischer/Marcel Krüger)*

- Changed the handling of the script key in harf mode to be more compatible with behaviour of the node mode. It now expects the name of a script that is actually in the font instead of a ISO 15924 script tag. See issue 117.
- Corrected a number of small typos and bugs in harf mode.

2.14 *New in version 3.10 (by Ulrike Fischer/Marcel Krüger)*

- The package has been moved to the github of the LaTeX3 Project and is now maintained officially by the LaTeX3 Project team.
- Code to use the harfbuzz library of luahbtex has been added. See the description of the harf mode.
- fonts in ttc-collections can now be indexed by name.
- To reduce the polution of the global lua enviroment a number of lua tables have been removed. Only the tables luaotfload, fonts and nodes have been kept there.
- The fontloader has been synched with the context files from 2019-10-29.

2.15 *New in version 3.00 (by Ulrike Fischer/Marcel Krüger)*

- Default Ignorable characters are now invisible by default (issue 63). This can be deactivated with the option `invisible`.

2.16 *New in version 2.99 (by Ulrike Fischer)*

- Code cleanup.
- The fontloader has been synched with the context files from 2019-08-11.

2.17 *New in version 2.98 (by Ulrike Fischer)*

breaking change!

- The handling of missing chars has been changed. In This version a missing char will insert the `/.notdef` char of the fonts (this is sometimes a space, sometimes a rectangle with a cross) and no longer simply ignore the glyph. This behaviour can be reverted by using `notdef=false` as font feature.
- The font feature `embolden` can now be used to fake a bold font.
- The fontloader has been synched with the context files from 2019-07-04.

2.18 *New in version 2.97 (by Ulrike Fischer)*

- the new generic fontloader improves the handling of large fonts (but some fonts still need a 64bit luatex version to create the font files).
- A number of small bug (also in luaotfload-tool) have been corrected, see the NEWS file for details.

2.19 *New in version 2.96 (by Ulrike Fischer)*

Incompatible change!

- In version 2.95 letterspacing was broken due to a change in the fontloader (issue 38). This has been repaired. At the same time a number of oddities and bugs in the letterspacing has been corrected. This can change existing documents. See page 15 for more information.
- A problem with the detection of bold fonts has been corrected (issue 41, pull request 42).

2.20 *New in version 2.95 (by Ulrike Fischer)*

- This version imports from context the generic fontloader in the version of 2019-01-28. Contrary to the last announcement, it still works with luatex 1.07. So updates will continue.
- The handling of the lucida-fonts had been improved (issue 33).
- tex-files are no longer misused as font fallbacks (issue 35).
- The resolver code has be refactorated (pull request 36).

2.21 *New in version 2.94 (by Ulrike Fischer)*

- This version imports from context the generic fontloader in the version of 2018-12-19. It is the last version that works with luatex 1.07 and texlive 2018. As context has moved to luatex 1.09 newer versions of the fontloader needs now this luatex version too. This means that until the texlive 2018 freeze there will be probably no update of luaotfload.
- This version changes the handling of the `mode` key. It no longer accepts only the values `base` and `node`, but can be used to load a font with an alternative font loader/renderer.

2.22 *New in version 2.93 (by Ulrike Fischer)*

Mainly internal clean up of the version info to allow automatic versioning.

2.23 *New in version 2.92 (by Ulrike Fischer)*

- Better devanagari support (issue #9).
- **Luaotfload** doesn't work when luatex is used with the option `--safer`. So it now aborts cleanly when the option is detected – but you still can get errors from fontspec later! (issue #12).
- The syntax `file:` for legacy font works again (issue #11).
- The fontloader has been synched with the newest context version from october, 18.

2.24 *New in version 2.91 (by Ulrike Fischer)*

This version mostly correct two bugs found in the previous fontloader: Glyphvariants weren't copied and pasted correctly. Glyphs encoded in the PUA couldn't be accessed anymore.

2.25 *New in version 2.9 (by Ulrike Fischer)*

On the one side there is not very much new in this version: The native components of Luaotfload are nearly unchanged. A few bugs have been corrected, the various files lists which loads the components of the font loader have been cleaned up.

On the other side there is a lot new:

- **Fontloader**
  The fontloader files imported from ConTEXt have been updated to the current version. This was necessary to make Luaotfload compatible with the coming LuaTEX 1.08/1.09. Compared to the previous version from february 2017 quite a number of things have changed. Most importantly the handling of arabic fonts has greatly improved. But this also means that changes in the output are possible.

- **Lualibs**
  The update of the fontloader files also required an update of the Lualibs package. This Luaotfload version needs version 2.6 of Lualibs.

- **Maintenance**
  As the current maintainer wasn't available and it was urgent to get a Luaotfload compatible with LuaTEX 1.08/1.09 maintenance has been transfered to Ulrike Fischer and Marcel Krüger. The package was maintained and developed at https://github.com/u-fischer/luaotfload.

- **Documentation**
  The core of documentation is nearly unchanged. I added this introduction. I recreated with the help of @marmot the graphic on 36. I updated the file lists. I imported as appendix pdf versions of the two man files which are part of the Luaotfload documentation.

3 Introduction

Font management and installation has always been painful with TEX. A lot of files are needed for one font (TFM, PFB, MAP, FD, VF), and due to the 8-Bit encoding each font is limited to 256 characters.

But the font world has evolved since the original TEX, and new typographic systems have appeared, most notably the so called *smart font* technologies like OpenType fonts (OTF).

These fonts can contain many more characters than TEX fonts, as well as additional functionality like ligatures, old-style numbers, small capitals, etc., and support more complex writing systems like Arabic and Indic[1] scripts.

OpenType fonts are widely deployed and available for all modern operating systems. As of 2013 they have become the de facto standard for advanced text layout.

However, until recently the only way to use them directly in the TEX world was with the XETEX engine.

Unlike XETEX, LuaTEX has no built-in support for OpenType or technologies other than the original TEX fonts.

Instead, it provides hooks for executing Lua code during the TEX run that allow implementing extensions for loading fonts and manipulating how input text is processed without modifying the underlying engine.

[1] Unfortunately, the default fontloader of luaotfload doesn't support many Indic scripts correctly. For these scripts it is recommended to use the harf mode along with the binary luahbtex.

This is where **luaotfload** comes into play: Based on code from ConTEXt, it extends LuaTEX with functionality necessary for handling **OpenType** fonts.

Additionally, it provides means for accessing fonts known to the operating system conveniently by indexing the metadata.

4 Thanks

Luaotfload is part of LuaLATEX, the community-driven project to provide a foundation for using the LATEX format with the full capabilites of the LuaTEX engine. As such, the distinction between end users, contributors, and project maintainers is intentionally kept less strict, lest we unduly personalize the common effort.

Nevertheless, the current maintainers would like to express their gratitude to Khaled Hosny, Akira Kakuto, Hironori Kitagawa and Dohyun Kim. Their contributions – be it patches, advice, or systematic testing – made the switch from version 1.x to 2.2 possible. Also, Hans Hagen, the author of the font loader, made porting the code to LATEX a breeze due to the extra effort he invested into isolating it from the rest of ConTEXt, not to mention his assistance in the task and willingness to respond to our suggestions.

5 Loading Fonts

luaotfload supports an extended font request syntax:

`\font\foo = {`⟨*prefix*⟩`:`⟨*font name*⟩`:`⟨*font features*⟩`}`⟨*TEX font features*⟩

The curly brackets are optional and escape the spaces in the enclosed font name. Alternatively, double quotes serve the same purpose. A selection of individual parts of the syntax are discussed below; for a more formal description see figure 1.

5.1 *Prefix – the* luaotfload *Way*

In **luaotfload**, the canonical syntax for font requests requires a *prefix*:

`\font\fontname =` ⟨*prefix*⟩`:`⟨*fontname*⟩...

where ⟨*prefix*⟩ is either `file:` or `name:`.[2] It determines whether the font loader should interpret the request as a *file name* or *font name*, respectively, which again influences how it will attempt to locate the font. Examples for font names are "Latin Modern Italic", "GFS Bodoni Rg", and "PT Serif Caption" – they are the human readable identifiers usually listed in drop-down menus and the like.[3] In order for fonts installed both in system

[2] **Luaotfload** also knows two further prefixes, `kpse:` and `my:`. A `kpse` lookup is restricted to files that can be found by **kpathsea** and will not attempt to locate system fonts. This behavior can be of value when an extra degree of encapsulation is needed, for instance when supplying a customized tex distribution.

The `my` lookup takes this a step further: it lets you define a custom resolver function and hook it into the **resolve_font** callback. This ensures full control over how a file is located. For a working example see the test in the luaotfload repo.

[3] Font names may appear like a great choice at first because they offer seemingly more intuitive identifiers in comparison to arguably cryptic file names: "PT Sans Bold" is a lot more descriptive than `PTS75F.ttf`. On the other hand, font names are quite arbitrary and there is no universal method to determine their meaning. While **luaotfload** provides fairly sophisticated heuristic to figure out a matching font style, weight, and optical size, it cannot be relied upon to work satisfactorily for all font files. For an in-depth analysis of the situation and how broken font names are, please refer to this post by Hans Hagen, the author of the font loader. If in doubt, use filenames. `luaotfload-tool` can perform the matching for you with the option `--find=<name>`, and you can use the file name it returns in your font definition.

locations and in your `texmf` to be accessible by font name, **luaotfload** must first collect the metadata included in the files. Please refer to section 8 below for instructions on how to create the database.

File names are whatever your file system allows them to be, except that that they may not contain the characters `(`, `:`, and `/`. As is obvious from the last exception, the `file:` lookup will not process paths to the font location – only those files found when generating the database are addressable this way. Continue below in the XeTeX section if you need to load your fonts by path. The file names corresponding to the example font names above are `lmroman12-italic.otf`, `GFSBodoni.otf`, and `PTZ56F.ttf`.

5.2 *Bracketed Lookups*

Bracketed lookups allow for arbitrary character content to be used in a definition. A simple bracketed request looks follows the scheme

```
\font\fontname = [⟨/path/to/file⟩]
```

Inside the square brackets, every character except for a closing bracket is permitted, allowing for arbitrary paths to a font file – including Windows style paths with UNC or drive letter prepended – to be specified. The **Luaotfload** syntax differs from XeTeX in that the subfont selector goes *after* the closing bracket:

```
\font\fontname = [⟨/path/to/file⟩](n)
```

Naturally, path-less file names are equally valid and processed the same way as an ordinary `file:` lookup.

5.3 *Compatibility*

In addition to the regular prefixed requests, **luaotfload** accepts loading fonts the XeTeX way. There are again two modes: bracketed and unbracketed. For the bracketed variety, see above, 5.2.

Unbracketed (or, for lack of a better word: *anonymous*) font requests resemble the conventional TeX syntax.

```
\font\fontname = ⟨font name⟩ ...
```

However, they have a broader spectrum of possible interpretations: before anything else, **luaotfload** attempts to load a traditional TeX Font Metric (TFM or OFM). If this fails, it performs a `path:` lookup, which itself will fall back to a `file:` lookup. Lastly, if none of the above succeeded, attempt to resolve the request as a `name:` lookup by searching the font index for ⟨*font name*⟩. The behavior of this "anonymous" lookup is configurable, see the configuation manpage for details.

Furthermore, **luaotfload** supports the slashed (shorthand) font style notation from XeTeX.

```
\font\fontname = ⟨font name⟩/⟨modifier⟩ ...
```

Currently, four style modifiers are supported: `I` for italic shape, `B` for bold weight, `BI` or `IB` for the combination of both. Other "slashed" modifiers are too specific to the XeTeX engine and have no meaning in LuaTeX.

5.4 *Examples*

5.4.1 Loading by File Name

For example, conventional TEX font can be loaded with a `file:` request like so:

```
\font  \lmromanten  = {file:ec-lmr10} at 10pt
```

The OpenType version of Janusz Nowacki's font *Antykwa Półtawskiego*[4] in its condensed variant can be loaded as follows:

```
\font  \apcregular  = file:antpoltltcond-regular.otf at 42pt
```

The next example shows how to load the *Porson* font digitized by the Greek Font Society using XETEX-style syntax and an absolute path from a non-standard directory:

```
\font  \gfsporson  = "[/tmp/GFSPorson.otf]" at 12pt
```

TrueType collection files (the extension is usually `.ttc`) contain more than a single font. In order to refer to these subfonts, the respective index or the respective PostScript font name may be added in parentheses after the file name.[5]

```
\font  \cambriamain  = "file:cambria.ttc(0)" at 10pt
\font  \cambriamath  = "file:cambria.ttc(1)" at 10pt
\font  \Cambriamain  = "file:cambria.ttc(Cambria)" at 10pt
\font  \Cambriamath  = "file:cambria.ttc(CambriaMath)" at 10pt
```

and likewise, requesting subfont inside a TTC container by path:

```
\font  \asanamain  = "[/home/typesetter/.fonts/math/asana.ttc](0):mode=node;+tlig" at 10pt
\font  \asanamath  = "[/home/typesetter/.fonts/math/asana.ttc](1):mode=base" at 10pt
```

5.4.2 Loading by Font Name

The `name:` lookup does not depend on cryptic filenames:

```
\font  \pagellaregular  = {name:TeX Gyre Pagella} at 9pt
```

A bit more specific but essentially the same lookup would be:

```
\font  \pagellaregular  = {name:TeX Gyre Pagella Regular} at 9pt
```

Which fits nicely with the whole set:

```
\font \pagellaregular     = {name:TeX Gyre Pagella Regular}    at 9pt
\font \pagellaitalic      = {name:TeX Gyre Pagella Italic}     at 9pt
\font \pagellabold        = {name:TeX Gyre Pagella Bold}       at 9pt
\font \pagellabolditalic  = {name:TeX Gyre Pagella Bolditalic} at 9pt
{\pagellaregular      foo bar baz\endgraf }
{\pagellaitalic       foo bar baz\endgraf }
```

[4] http://jmn.pl/antykwa-poltawskiego/, also available in in TEX Live.

[5] Incidentally, this syntactical detail also prevents one from loading files that end in balanced parentheses.

```
{\pagellabold         foo bar baz\endgraf }
{\pagellabolditalic   foo bar baz\endgraf }
...
```

5.4.3 Modifiers

If the entire *Iwona* family[6] is installed in some location accessible by **luaotfload**, the regular shape can be loaded as follows:

```
\font  \iwona  = Iwona at 20pt
```

To load the most common of the other styles, the slash notation can be employed as shorthand:

```
\font  \iwonaitalic      = Iwona/I    at 20pt
\font  \iwonabold        = Iwona/B    at 20pt
\font  \iwonabolditalic  = Iwona/BI   at 20pt
```

which is equivalent to these full names:

```
\font  \iwonaitalic      = "Iwona Italic"       at 20pt
\font  \iwonabold        = "Iwona Bold"         at 20pt
\font  \iwonabolditalic  = "Iwona BoldItalic"   at 20pt
```

6 Font features

Font features are the second to last component in the general scheme for font requests:

```
\font\foo = "⟨prefix⟩:⟨font name⟩:⟨font features⟩⟨TEX font features⟩"
```

If style modifiers are present (XeTeX style), they must precede ⟨*font features*⟩.

The element ⟨*font features*⟩ is a semicolon-separated list of feature tags[7] and font options. Prepending a font feature with a + (plus sign) enables it, whereas a - (minus) disables it. For instance, the request

```
\font  \test  = LatinModernRoman:+clig;-kern
```

activates contextual ligatures (`clig`) and disables kerning (`kern`). Alternatively the options `true` or `false` can be passed to the feature in a key/value expression. The following request has the same meaning as the last one:

```
\font  \test  = LatinModernRoman:clig=true;kern=false
```

Furthermore, this second syntax is required should a font feature accept other options besides a true/false switch. For example, *stylistic alternates* (`salt`) are variants of given glyphs. They can be selected either explicitly by supplying the variant index (starting from one), or randomly by setting the value to, obviously, `random`.

```
\font  \librmsaltfirst  = LatinModernRoman:salt=1
```

[6] http://jmn.pl/kurier-i-iwona/, also in TeX Live.

[7] Cf. http://www.microsoft.com/typography/otspec/featurelist.htm.

6.1 *Basic font features*

- **mode**
  luaotfload had three OpenType processing *modes*: base, node and harf.

  base mode works by mapping OpenType features to traditional TEX ligature and kerning mechanisms. Supporting only non-contextual substitutions and kerning pairs, it is the slightly faster, albeit somewhat limited, variant. node mode works by processing TEX's internal node list directly at the Lua end and supports a wider range of OpenType features. The downside is that the intricate operations required for node mode may slow down typesetting especially with complex fonts and it does not work in math mode.

  By default luaotfload is in node mode, and base mode has to be requested where needed, e. g. for math fonts.

  harf mode is new in version 3.1 and needs the new luahbtex engine (the mode is ignored if luahbtex is not used). With it is possible to render a font using the harfbuzz library in-built in the new engine. harf mode improves greatly the rendering of indic and arabic scripts and is highly recommended for such scripts.

  When using harf mode it is required to set also the script correctly.

```
\font \burmesefont ={file:NotoSerifMyanmar-Regular.ttf:mode=harf;script=mym2;}
\font \devafont ={file:NotoSansDevanagari-Regular.ttf:mode=harf;script=dev2;}
\font \banglafont ={name:Noto Sans Bengali:mode=harf;script=ben2;}
\font \tibetanfont ={name:Noto Serif Tibetan:mode=harf;script=tibt;}
```

  မ္မတီခေါင်းလောင်းကြီ
  क्रर्कि
  কণ্যা এখন কি করিবে
  སྐྱེས་ཙམ་ཉིད་ནས་ཆེ་མཐོངས་དང་།

  It is possible to call other font renderers with the mode key. A simple example with a plain reader can be found at https://github.com/latex3/luaotfload/pull/26#issuecomment-437716326.

- **shaper**
  If luahbtex and harf mode are used it is possible to specify a shaper, like graphite2 or ot (open type).

```
\pardir  TRT\textdir  TRT
\font \test ={file:AwamiNastaliq-Regular.ttf:mode=harf;shaper=ot}
```

  ارش ل ی م ئ ے

```
\pardir  TRT\textdir  TRT
\font \test ={file:AwamiNastaliq-Regular.ttf:mode=harf;shaper=graphite2}
```

اُرْشَلِيمَ

- **script**
  An OpenType script tag;[8] the default value is `dflt`. Some fonts, including very popular ones by foundries like Adobe, do not assign features to the `dflt` script, in which case the script needs to be set explicitly.

- **language**
  An OpenType language system identifier,[9] defaulting to `dflt`.

- **color**
  A font color, defined as a triplet of two-digit hexadecimal RGB values, with an optional fourth value for transparency (where `00` is completely transparent and `FF` is opaque).

  For example, in order to set text in semitransparent red:

```
\font  \test  = "Latin Modern Roman:color=FF0000BB"
```

NEW in v3.12!

Experimental! The **color** key has been extended to accept a table with color declarations of (output) glyphs. For example

```
\directlua {
  luaotfload.add_colorscheme("myscheme",
  {
    ["00FFFF30"] = {"default"},
    ["FF0000"] = {"kabeng","ebeng"},
    ["00FF00"] = {"ivowelsignbeng"},
    ["0000FF"] = {369}            })
  }
```

The keys in such a table are like above RGB colors with an optional transparency setting. The values are either lists of glyph names or GID numbers. Both types are font dependant! Not every font use the same glyph names (or even glyph names at all). GID number are font specific anyway. The GID can be found by looking up the `["index"]` entry in the lua file of a font.

Such a colorscheme can then be used like this:

```
\font \test ={name:Noto Sans Bengali:mode=harf;script=bng2;color=myscheme}
```

and then would give this output:

কণ্যা এখন কি করিবে কিন্দ্র

- **axis&instance**

NEW in v3.15!

Experimental! Support for OpenType variable fonts. *Varible fonts are only sup-*

[8] See http://www.microsoft.com/typography/otspec/scripttags.htm for a list of valid values. For scripts derived from the Latin alphabet the value `latn` is good choice.

[9] Cf. http://www.microsoft.com/typography/otspec/languagetags.htm.

ported in base *and* node *mode, not in* harf *mode.*

To specify the parameters of a variable font, you can either specify a predefined instance of the font by passing the associated "subfamily" name to instance or parameters for individual axis can be provided using the axis feature. You can *not* use instance and axis together.

For example (needs the variable Fraunces font installed)

```
\def \fraunces #1#2{             \font \varfont  = "Fraunces/B:mode=node;#1;" at #2pt\varfont
}
\fraunces {axis={wght=Regular}}{10}Regular font\par
\fraunces {axis={wght=Black}}{10}Black variant (aka. very bold)\par
\fraunces {axis={wght=Black,opsz=10}}{10}Black again, but with
correct optical size\par
\fraunces {axis={weight=100,opsz=10}}{10}Let's try giving axis values
numerically\par
\fraunces {instance=semibold}{10}A semi-bold one given as
a instance
(Corresponding to
\verb |axis={opsz=144,wght=600,SOFT=100,WONG=1}|)\par
```

Regular font

Black variant (aka. very bold)

Black again, but with correct optical size

Let's try giving axis values numerically

A semi-bold one given as a instance
(Corresponding to axis={opsz=144,wght=600,SOFT=100,WONG=1}**)**

- **embolden**
  A factor, defined as a decimal number.

  For example

```
\font \test  = "Latin Modern Roman:mode=node;embolden=2;"
```

  Dies is not bold. Dies is a faked bold font.

- **kernfactor & letterspace**
  Define a font with letterspacing (tracking) enabled. In luaotfload, letterspacing is implemented by inserting additional kerning between glyphs.

  This approach is derived from and still quite similar to the *character kerning* (\setcharacterkerning / \definecharacterkerning & al.) functionality of Context, see the file typo-krn.lua there. The main difference is that luaotfload does not use LuaTEX attributes to assign letterspacing to regions, but defines virtual letterspaced versions of a font.

The option kernfactor accepts a numeric value that determines the letterspacing factor to be applied to the font size. E. g. a kern factor of 0.42 applied to a 10 pt font results in 4.2 pt of additional kerning applied to each pair of glyphs.

NEW in v2.96! Spaces between words are now stretched too. This is consistent with the XeTeX behaviour (and the amount of stretching should be similar). This naturally changes the output of a document. In case you want the old behaviour back use

```
\directlua {luaotfload.letterspace.keepwordspacing = true}
```

The difference between both options is obvious:

N e w : h e l l o w o r l d

O l d : h e l l o w o r l d

NEW in v2.96! Ligatures are no longer split into their component glyphs. This change too make the luaotfload more compatible with XeTeX. It also makes it much easier to activate or deactivate ligature sets in letterspaced fonts. If you want to split ligatures, you should deactivate as you would do it with a not-letterspaced font, e.g. with the fontspec Ligatures option, or the low-level -liga and similar.

W i t h s t a n d a r d l i g a t u r e s : fi – ff

O n l y w i t h t l i g : fi – ff

N o l i g a t u r e s : fi -- ff

For compatibility with XeTeX an alternative letterspace option is supplied that interprets the supplied value as a *percentage* of the font size but is otherwise identical to kernfactor. Consequently, both definitions in below snippet yield the same letterspacing width:

```
\font  \iwonakernedA  = "file:Iwona-Regular.otf:kernfactor=0.125"
\font  \iwonakernedB  = "file:Iwona-Regular.otf:letterspace=12.5"
```

The microtype package uses a special implementation of letterspacing, and the commands \lsstyle and \textls are not affected by these changes.

Setting the ligatures with the font options is the recommended way, to activate or deactivate them. In case of special requirements specific pairs of letters and ligatures may be exempt from letterspacing by defining the Lua functions keeptogether and keepligature, respectively, inside the namespace luaotfload.letterspace. Both functions are called whenever the letterspacing callback encounters an appropriate node or set of nodes. If they return a true-ish value, no extra kern is inserted at the current position. keeptogether receives a pair of consecutive glyph nodes in order of their appearance in the node list. keepligature receives a single node which can be analyzed into components. (For details refer to the *glyph nodes* section in the LuaTeX reference manual.) The implementation of both functions is left entirely to the user.

- **protrusion & expansion**
  These keys control microtypographic features of the font, namely *character pro-*

trusion and *font expansion*. Their arguments are names of Lua tables that contain values for the respective features.[10] For both, only the set **default** is predefined.

For example, to define a font with the default protrusion vector applied[11]:

```
\font  \test  = LatinModernRoman:protrusion=default
```

- **invisible**
  Default Ignorable characters are control characters that should be invisible by default even if the font has glyphs for them. Since version 3.0 luaotfload makes them invisible, this can be switch on and off with the `invisible`. By default it is on.

  For example

```
\font \amiri ={file:amiri-regular.ttf} at 20pt \amiri
\char "200D□\char "200D
```

ي

```
\font \amiri ={file:amiri-regular.ttf:-invisible;} at 20pt \amiri
\char "200D□\char "200D
```

أيأ

- **multiscript**
  New in 3.12 – experimental

  In fonts many shaping rules are implemented only for specific scripts and so you get correct typesetting only if the **script** feature is correctly set. This means that to write a text which uses more than one script you have to declare a font for each script and switch fonts even if the font contains glyphs for all scripts. **multiscript** tries to help here. The feature is experimental and bound to change. Feedback is welcome but you use it at your risk.

  multiscript allows you to declare fonts for various script. The value is either **auto** described below, or a name which has been previously declared or a combination of both. An example for such a named multiscript could look like this (the colors are only for demonstration):

```
\directlua {
```

[10] For examples of the table layout please refer to the section of the file `luaotfload-fonts-ext.lua` where the default values are defined. Alternatively and with loss of information, you can dump those tables into your terminal by issuing

```
\directlua {inspect(fonts.protrusions.setups.default)
            inspect(fonts.expansions.setups.default)}
```

at some point after loading `luaotfload.sty`.

[11] You also need to set `pdfprotrudechars=2` and `pdfadjustspacing=2` to activate protrusion and expansion, respectively. See the pdfTEX manualfor details.

```
luaotfload.add_multiscript
  ("cyrlgrekbeng",
   {
     Cyrl = "DejaVuSans:mode=node;script=cyrl;color=FF0000;",
     Grek = "texgyreheros:mode=harf;script=grek;color=0000FF;",
     Beng = "NotoSansBengali:mode=harf;script=bng2;color=00FF00;"
   }
  )
 }
```

cyrlgrekbeng is the name of the multiscript (the name is case insensitive). The keys are ISO language tags (not open type tags!). They are case insensitive too: the example uses an uppercase letter for ISO tags to differentiate them from script tags. The values are font declarations.

The multiscript can then be used in a font like this:

```
\font \test ={name:DejaVuSans:mode=node;multiscript=cyrlgrekbeng;}
```

This would lead to this output:

„á123!?“ „π123!?“ „a!?“ „Б123!?“ a „কি123“

It shows that fonts are switched with the scripts.

Be aware of the following drawbacks:

- Quite a lot chars can and should be used with more than one script, they belong to the Common or Inherited class. Examples are punctuation chars, digits, accents but also emoji. Currently these chars follow the active script. That's why the digits are all typeset with a different font, the accent over the pi is different to the one over the a, and why the opening quote is sometimes different to the closing quote. It is clear that some tools to force a script (and so a font) locally and globally for such chars are needed.
- multiscript doesn't change hyphenation patterns or other language or script related features.
- Language packages like babel or polyglossia have code to change the script too which could interfere or clash. This hasn't been tested yet.
- multiscript can slow down the compilation.

It is possible to use the value auto with multiscript. luaotfload will then switch the script if it detects a char belonging to another script (and if the font support this script). This can be useful for fonts supporting more than one script or when using the fallback key described below.

It is also possible to combine auto with a named multiscript with the syntax multiscript=auto+name. The rules of the named multiscript will in such cases take precedence and auto used only for other scripts.

New in 3.12 – experimental

- **fallback**
  This allows you to define a chain of fonts which are used if glyphs are missing in the main font. It works only for text fonts, not for math fonts set with the unicode-math package. The feature is experimental and bound to change. Feedback is welcome but you use it at your risk. For example

```
\directlua
 {luaotfload.add_fallback
  ("myfallback",
    {
     "DejaVuSans:mode=harf;script=grek;color=FF0000;",
     "cmuserif:mode=node;script=cyrl;color=00FF00;",
     "NotoSansBengali:mode=harf;script=bng2;color=0000FF;",
     "NotoColorEmoji:mode=harf;"
    }
  )
 }
```

  This fallback can then be used e.g. like this:

```
\font \test ={name:LatinModernRoman:mode=node;fallback=myfallback;}
```

  1234 á π́ a!? π123!? a БЊ123!? a 😀 🦆 কি a „π“ a „Б“

  Interesting points in the output are

  – The accent over the pi, the digits and the quotes are all from the base font. Only missing glyphs are from the fallback.
  – The cyrillic is printed with the DejaVu font, despite the fact that is sets the script to grek and that the next font in the fallback chain would fit better.
  – The duck emoji is from the Noto font, while the face is from DejaVu as it comes first in the chain.

  The fallback can be combined with the multiscript. For example

```
\font \test ={name:LatinModernRoman:mode=node;fallback=myfallback;multiscript=auto;}
```

  1234 á π́ a!? π123!? a БЊ123!? a 😀 🦆 কি a „π“ a „Б“

  Now the accent over the pi is better. The digits after the pi and the closing quote use the DejaVu font. The digits after the cyrillic use the Latin Modern font because of an interesting “feature” of this font: It claims to know the cyrl script despite the fact that it doesn’t contain any cyrillic glyphs.

  fallback can be nested: fonts in the fallback table can refer to another fallback table.

  As with the multiscript key more control over the used glyph and script in edge cases will be needed.

- **upper/lower**

New in 3.16

The font features upper and lower can be used to map the text of a font to upper or lowercase before displaying it.

By default the casemapping is automatically tailored based on the active language font feature. This automatic tailoring can be overwritten by passing a two letter language tag to the upper or lower feature. In addition to ordinary language tags, the tags el-x-iota, hy-x-yiwn and de-x-eszett can be used. el-x-iota adapts the greek rules for uppercasing to keep iota subscripts as combining characters which sufficiently well behaving fonts can render in appropriate ways while the default for greek is to convert these into capital iotas. de-x-eszett adapts uppercasing for german to use the capital eszett instead of uppercasing the eszett to double capital s.

```
\font \test ={kpse:LinLibertine_R.otf:mode=node;+upper;}
\test  Grüße
\font \test ={kpse:LinLibertine_R.otf:mode=node;upper=de-x-eszett;}
\test  Grüße
```

GRÜSSE GRÜßE

```
\font \test ={kpse:LinLibertine_R.otf:mode=node;script=grek;+upper;}
No specified language: {\test  □□□□□□□□□□ □□□□}\par
\font \test ={kpse:LinLibertine_R.otf:mode=node;script=grek;language=ell;+upper;}
Greek uppercasing: {\test  □□□□□□□□□□ □□□□}\par
\font \test ={kpse:LinLibertine_R.otf:mode=node;script=grek;language=ell;+upper=el-x-iota;}
Greek variant: {\test  □□□□□□□□□□ □□□□}\par
```

No specified language: Ε᾽ΙΣΕΝΈΓΚΉΙΣ Ἡ ΜΑΣ

Greek uppercasing: ΕΙΣΕΝΕΓΚΗΙΣ ΗΜΑΣ

Greek variant: ΕΙΣΕΝΕΓΚΗιΣ ΗΜΑΣ

- **Variable fonts**

New in 3.16

When the node shaper is used, experimental support for OpenType variable fonts has been added. To use them, set the font feature axis to a comma separated list of axis names and values. (E.g. axis=weight=600) The supported axis names and value range depend on the font (see page 20). The following listing shows an example with the Source Code Variable font:

```
\documentclass{article}
 \DeclareFontFamily{TU}{sourcecode-variable}{}
 \newcommand\DeclareSourceVariable[2]{%
   \DeclareFontShape{TU}{sourcecode-variable}{#1}{n}{%
     <-> \UnicodeFontFile{SourceCodeVariable-Roman.otf}
         {\UnicodeFontTeXLigatures axis={weight=#2};}%
   }{}%
   \DeclareFontShape{TU}{sourcecode-variable}{#1}{it}{%
```

```
    <-> \UnicodeFontFile{SourceCodeVariable-Italic.otf}
       {\UnicodeFontTeXLigatures axis={weight=#2};}%
  }{}%
}
\DeclareSourceVariable{ul}{200}
\DeclareSourceVariable{el}{250}
\DeclareSourceVariable{l}{300}
\DeclareSourceVariable{sl}{350}
\DeclareSourceVariable{m}{400}
\DeclareSourceVariable{sb}{500}
\DeclareSourceVariable{b}{600}
\DeclareSourceVariable{eb}{700}
\DeclareSourceVariable{ub}{900}
\begin{document}
\fontfamily{sourcecode-variable}\selectfont
\fontseries{ul}\selectfont a\textit{b}
\fontseries{el}\selectfont c\textit{d}
\fontseries{l}\selectfont e\textit{f}
\fontseries{sl}\selectfont g\textit{h}
\fontseries{m}\selectfont i\textit{j}
\fontseries{sb}\selectfont k\textit{l}
\fontseries{b}\selectfont m\textit{n}
\fontseries{eb}\selectfont o\textit{p}
\fontseries{ub}\selectfont q\textit{r}
\end{document}
```

6.2 *Non-standard font features*

luaotfload adds a number of features that are not defined in the original OpenType specification, most of them aiming at emulating the behavior familiar from other TEX engines. Currently (2014) there are three of them:

- **anum**
  Substitutes the glyphs in the ASCII number range with their counterparts from eastern Arabic or Persian, depending on the value of language.

- **tlig**
  Applies legacy TEX ligatures[12]:

“	‘‘	”	’’
‘	‘	’	’
”	”	–	--
—	---	¡	!‘
¿	?‘		

- **itlc**
  Computes italic correction values (active by default).

[12] These contain the feature set trep of earlier versions of luaotfload.
Note to XETEX users: this is the equivalent of the assignment mapping=text-tex using XETEX's input remapping feature.

7 Combining fonts

Beside the new keys multiscript and fallback described earlier Version 2.7 and later support another method to combine characters from multiple fonts into a single virtualized one. This requires that the affected fonts be loaded in advance as well as a special *request syntax*. Furthermore, this allows to define *fallback fonts* to supplement fonts that may lack certain required glyphs.

Combinations are created by defining a font using the combo: prefix.

7.1 *Fallbacks*

For example, the Latin Modern family of fonts does, as indicated in the name, not provide Cyrillic glyphs. If Latin script dominates in the copy with interspersed Cyrillic, a fallback can be created from a similiar looking font like Computer Modern Unicode, taking advantage of the fact that it too derives from Knuth's original Computer Modern series:

```
\input  luaotfload.sty
\font  \lm   = file:lmroman10-regular.otf:mode=base
\font  \cmu  = file:cmunrm.otf:mode=base
\font  \lmu  = "combo: 1->\fontid \lm ; 2->\fontid \cmu ,fallback"
\lmu  Eh bien, mon prince. Gênes et Lueques ne sont plus que des
     apanages, des □□□□□□□□, de la famille Buonaparte.
\bye
```

As simple as this may look on the first glance, this approach is entirely inappropriate if more than a couple letters are required from a different font. Because the combination pulls nothing except the glyph data, all of the important other information that constitute a proper font – kerning, styles, features, and suchlike – will be missing.

7.2 *Combinations*

Generalizing the idea of a *fallback font*, it is also possible to pick definite sets of glyphs from multiple fonts. On a bad day, for instance, it may be the sanest choice to start out with EB Garamond italics, typeset all decimal digits in the bold italics of GNU Freefont, and tone down the punctuation with extra thin glyphs from Source Sans:

```
\def  \feats      {-tlig;-liga;mode=base;-kern}
\def  \fileone    {EBGaramond12-Italic.otf}
\def  \filetwo    {FreeMonoBoldOblique.otf}
\def  \filethree  {SourceSansPro-ExtraLight.otf}
\input  luaotfload.sty
\font  \one    = file:\fileone   :\feats
\font  \two    = file:\filetwo   :\feats
\font  \three  = file:\filethree :\feats
\font  \onetwothree  = "combo:  1 -> \fontid \one ;
                              2 -> \fontid \two ,   0x30-0x39;
                              3 -> \fontid \three , 0x21*0x3f; "
{\onetwothree  \TeX –0123456789–?!}
\bye
```

Despite the atrocious result, the example demonstrates well the syntax that is used to specify ranges and fonts. Fonts are being referred to by their internal index which can be obtained by passing the font command into the \fontid macro, e. g. \fontid\one, after a font has been defined. The first component of the combination is the base font which will be extended by the others. It is specified by the index alone.

All further fonts require either the literal fallback or a list of codepoint definitions to be appended after a comma. The elements of this list again denote either single codepoints like 0x21 (referring to the exclamation point character) or ranges of codepoints (0x30-0x39). Elements are separated by the ASCII asterisk character (*). The characters referenced in the list will be imported from the respective font, if available.

8 Font names database

As mentioned above, **luaotfload** keeps track of which fonts are available to LuaTEX by means of a *database*. This allows referring to fonts not only by explicit filenames but also by the proper names contained in the metadata which is often more accessible to humans.[13]

When **luaotfload** is asked to load a font by a font name, it will check if the database exists and load it, or else generate a fresh one. Should it then fail to locate the font, an update to the database is performed in case the font has been added to the system only recently. As soon as the database is updated, the resolver will try and look up the font again, all without user intervention. The goal is for **luaotfload** to act in the background and behave as unobtrusively as possible, while providing a convenient interface to the fonts installed on the system.

Generating the database for the first time may take a while since it inspects every font file on your computer. This is particularly noticeable if it occurs during a typesetting run. In any case, subsequent updates to the database will be quite fast.

8.1 luaotfload-tool

It can still be desirable at times to do some of these steps manually, and without having to compile a document. To this end, **luaotfload** comes with the utility luaotfload-tool that offers an interface to the database functionality. Being a Lua script, there are two ways to run it: either make it executable (chmod +x on unixoid systems) or pass it as an argument to texlua.[14] Invoked with the argument --update it will perform a database update, scanning for fonts not indexed.

```
luaotfload-tool --update
```

Adding the --force switch will initiate a complete rebuild of the database.

```
luaotfload-tool --update --force
```

[13] The tool otfinfo (comes with TEX Live), when invoked on a font file with the -i option, lists the variety of name fields defined for it.

[14] Tests by the maintainer show only marginal performance gain by running with Luigi Scarso's LuajitTEX, which is probably due to the fact that most of the time is spent on file system operations.

Note: On MS Windows systems, the script can be run either by calling the wrapper application luaotfload-tool.exe or as texlua.exe luaotfload-tool.lua.

Table 1: List of paths searched for each supported operating system.

8.2 *Search Paths*

luaotfload scans those directories where fonts are expected to be located on a given system. On a Linux machine it follows the paths listed in the **Fontconfig** configuration files; consult `man 5 fonts.conf` for further information. On **Windows** systems, the standard location is `Windows\\Fonts`, while **Mac OS X** requires a multitude of paths to be examined. The complete list is is given in table 1. Other paths can be specified by setting the environment variable `OSFONTDIR`. If it is non-empty, then search will be extended to the included directories.

Windows	`% WINDIR%\ Fonts`
Linux	`/usr/local/etc/fonts/fonts.conf` and `/etc/fonts/fonts.conf`
Mac	`~/Library/Fonts`, `/Library/Fonts`, `/System/Library/Fonts`, and `/Network/Library/Fonts`

8.3 *Querying from Outside*

`luaotfload-tool` also provides rudimentary means of accessing the information collected in the font database. If the option `--find=`*name* is given, the script will try and search the fonts indexed by **luaotfload** for a matching name. For instance, the invocation

```
luaotfload-tool  --find="Iwona Regular"
```

will verify if “Iwona Regular” is found in the database and can be readily requested in a document.

If you are unsure about the actual font name, then add the `-F` (or `--fuzzy`) switch to the command line to enable approximate matching. Suppose you cannot precisely remember if the variant of **Iwona** you are looking for was “Bright” or “Light”. The query

```
luaotfload-tool  -F --find="Iwona Bright"
```

will tell you that indeed the latter name is correct.

Basic information about fonts in the database can be displayed using the `-i` option (`--info`).

```
luaotfload-tool  -i --find="Iwona Light Italic"
```

The meaning of the printed values is described in section 4.4 of the LuaTEX reference manual.[15]

For a much more detailed report about a given font try the `-I` option instead (`--inspect`).

```
luaotfload-tool  -I --find="Iwona Light Italic"
```

[15] In TEX Live: `texmf-dist/doc/luatex/base/luatexref-t.pdf`.

`luaotfload-tool --help` will list the available command line switches, including some not discussed in detail here. For a full documentation of **luaotfload-tool** and its capabilities refer to the manpage (`man 1 luaotfload-tool`).[16]

8.4 *Blacklisting Fonts*

Some fonts are problematic in general, or just in LuaTEX. If you find that compiling your document takes far too long or eats away all your system's memory, you can track down the culprit by running `luaotfload-tool -v` to increase verbosity. Take a note of the *filename* of the font that database creation fails with and append it to the file `luaotfload-blacklist.cnf`.

A blacklist file is a list of font filenames, one per line. Specifying the full path to where the file is located is optional, the plain filename should suffice. File extensions (`.otf`, `.ttf`, etc.) may be omitted. Anything after a percent (`\%`) character until the end of the line is ignored, so use this to add comments. Place this file to some location where the **kpse** library can find it, e. g. `texmf-local/tex/luatex/luaotfload` if you are running TEX Live,[17] or just leave it in the working directory of your document. **luaotfload** reads all files named `luaotfload-blacklist.cnf` it finds, so the fonts in `./luaotfload-blacklist.cnf` extend the global blacklist.

Furthermore, a filename prepended with a dash character (-) is removed from the blacklist, causing it to be temporarily whitelisted without modifying the global file. An example with explicit paths:

```
/Library/Fonts/GillSans.ttc  -/Library/Fonts/Optima.ttc
```

9 The Fontloader

9.1 *Overview*

To a large extent, **luaotfload** relies on code originally written by Hans Hagen for the ConTEXt format. It integrates the font loader, written entirely in Lua, as distributed in the LuaTEX-Fonts package. The original Lua source files have been combined using the ConTEXt packaging library into a single, self-contained blob. In this form the font loader depends only on the **lualibs** package and requires only minor adaptions to integrate into **luaotfload**.

The guiding principle is to let ConTEXt/LuaTEX-Fonts take care of the implementation, and update the imported code as frequently as necessary. As maintainers, we aim at importing files from upstream essentially *unmodified*, except for renaming them to prevent name clashes. This job has been greatly alleviated since the advent of LuaTEX-Fonts, prior to which the individual dependencies had to be manually spotted and extracted from the ConTEXt source code in a complicated and error-prone fashion.

9.2 *Contents and Dependencies*

Below is a commented list of the files distributed with **luaotfload** in one way or the other. See see the figure on page 36 for a graphical representation of the dependencies. Through the script `mkimport` a ConTEXt library is invoked to create the **luaotfload** fontloader as a

[16] Or see `luaotfload-tool.rst` in the source directory.

[17] You may have to run `mktexlsr` if you created a new file in your `texmf` tree.

merged (amalgamated) source file.[18] This file constitutes the "default fontloader" and is part of the **luaotfload** package as `fontloader-YY-MM-DD.lua`, where the uppercase letters are placeholders for the build date. A companion to it, `luatex-basics-gen.lua` (renamed to `fontloader-basics-gen.lua` in **luaotfload**) must be loaded beforehand to set up parts of the environment required by the ConTEXt libraries. During a TEX run, the fontloader initialization and injection happens in the module `luaotfload-init.lua`. Additionally, the "reference fontloader" as imported from LuaTEX-Fonts is provided as the file `fontloader-reference.lua`. This file is self-contained in that it packages all the auxiliary Lua libraries too, as Luaotfload did up to the 2.5 series; since that job has been offloaded to the **Lualibs** package, loading this fontloader introduces a certain code duplication.

A number of *Lua utility libraries* are not part of the **luaotfload** fontloader, contrary to its equivalent in LuaTEX-Fonts. These are already provided by the **lualibs** and have thus been omitted from the merge.[19]

- `l-lua.lua`
- `l-lpeg.lua`
- `l-function.lua`
- `l-string.lua`
- `l-table.lua`
- `l-io.lua`
- `l-file.lua`
- `l-boolean.lua`
- `l-math.lua`
- `l-unicode.lua`
- `util-str.lua`
- `util-fil.lua`

The reference fontloader is home to several Lua files that can be grouped twofold as below:

- The *font loader* itself. These files have been written for LuaTEX-Fonts and they are distributed along with **luaotfload** so as to resemble the state of the code when it was imported. Their purpose is either to give a slightly aged version of a file if upstream considers latest developments for not yet ready for use outside Context; or, to install placeholders or minimalist versions of APIs relied upon but usually provided by parts of Context not included in the fontloader.
  - `luatex-basics-nod.lua`
  - `luatex-basics-chr.lua`
  - `luatex-fonts-mis.lua`
  - `luatex-fonts-enc.lua`
  - `luatex-fonts-syn.lua`
  - `luatex-fonts-tfm.lua`
  - `luatex-fonts-def.lua`
  - `luatex-fonts-ext.lua`
  - `luatex-fonts-lig.lua`
  - `luatex-fonts-gbn.lua`
  - `luatex-fonts.lua`
- Code related to *font handling and node processing*, taken directly from ConTEXt.

[18] In ConTEXt, this facility can be accessed by means of a script which is integrated into `mtxrun` as a subcommand. Run `mtxrun --script package --help` to display further information. For the actual merging code see the file `util-mrg.lua` that is part of ConTEXt.

[19] Faithful listeners will remember the pre-2.6 era when the fontloader used to be integrated as-is which caused all kinds of code duplication with the pervasive **lualibs** package. This conceptual glitch has since been amended by tightening the coupling with the excellent ConTEXt toolchain.

- data-con.lua
- font-ini.lua
- font-con.lua
- font-cid.lua
- font-map.lua
- font-vfc.lua
- font-otr.lua
- font-cff.lua
- font-ttf.lua
- font-dsp.lua
- font-oti.lua
- font-ott.lua
- font-otl.lua
- font-oto.lua
- font-otj.lua
- font-oup.lua
- font-ota.lua
- font-ots.lua
- font-otc.lua
- font-osd.lua
- font-ocl.lua
- font-onr.lua
- font-one.lua
- font-afk.lua
- font-lua.lua
- font-def.lua
- font-shp.lua
- font-imp-tex.lua
- font-imp-ligatures.lua
- font-imp-italics.lua
- font-imp-effects.lua

As an alternative to the merged file, **Luaotfload** may load individual unpackaged Lua libraries that come with the source, or even use the files from Context directly. Thus if you prefer running bleeding edge code from the ConTEXt beta, choose the `context` fontloader via the configuration file (see sections 10 and 9.3 below).

Also, the merged file at some point loads the Adobe Glyph List from a Lua table that is contained in `luaotfload-glyphlist.lua`, which is automatically generated by the script `mkglyphlist`.[20] There is a make target **glyphs** that will create a fresh glyph list so we don't need to import it from ConTEXt any longer.

In addition to these, **luaotfload** requires a number of files not contained in the merge. Some of these have no equivalent in LuaTEX-Fonts or ConTEXt, some were taken unmodified from the latter.

- `luaotfload-features.lua` – font feature handling; incorporates some of the code from `font-otc` from ConTEXt;
- `luaotfload-configuration.lua` – handling of `luaotfload.conf(5)`.
- `luaotfload-log.lua` – overrides the ConTEXt logging functionality.
- `luaotfload-loaders.lua` – registers readers in the fontloader for various kinds of font formats
- `luaotfload-parsers.lua` – various LPEG-based parsers.
- `luaotfload-database.lua` – font names database.
- `luaotfload-resolvers.lua` – file name resolvers.
- `luaotfload-colors.lua` – color handling.

[20] See `luaotfload-font-enc.lua`. The hard-coded file name is why we have to replace the procedure that loads the file in `luaotfload-init.lua`.

- `luaotfload-auxiliary.lua` – access to internal functionality for package authors (proposals for additions welcome).
- `luaotfload-letterspace.lua` – font-based letterspacing.
- `luaotfload-filelist.lua` – data about the files in the package.

9.3 *Packaging*

The fontloader code is integrated as an isolated component that can be switched out on demand. To specify the fontloader you wish to use, the configuration file (described in section 10) provides the option `fontloader`. Its value can be one of the identifiers `default` or `reference` (see above, section 9.2) or the name of a file somewhere in the search path of LuaTEX. This will make **Luaotfload** locate the ConTEXt source by means of **kpathsea** lookups and use those instead of the merged package. The parameter may be extended with a path to the ConTEXt `texmf`, separated with a colon:

```
[run]
  fontloader = context:~/context/tex/texmf-context
```

This setting allows accessing an installation – e. g. the standalone distribution or a source repository – outside the current TEX distribution.

Like the **Lualibs** package, the fontloader is deployed as a *merged package* containing a series of Lua files joined together in their expected order and stripped of non-significant parts. The `mkimport` utility assists in pulling the files from a ConTEXt tree and packaging them for use with **Luaotfload**.The state of the files currently in **Luaotfload**'s repository can be queried:

```
./scripts/mkimport news
```

The subcommand for importing takes the prefix of the desired ConTEXt **texmf** as an optional argument:

```
./scripts/mkimport import ~/context/tex/texmf-context
```

Whereas the command for packaging requires a path to the *package description file* and the output name to be passed.

```
./scripts/mkimport package fontloader-custom.lua
```

From the toplevel makefile, the targets `import` and `package` provide easy access to the commands as invoked during the **Luaotfload** build process.[21] These will call `mkimport` script with the correct parameters to generate a datestamped package. Whether files have been updated in the upstream distribution can be queried by `./scripts/mkimport news`. This will compare the imported files with their counterparts in the ConTEXt distribution and report changes.

[21] *Hint for those interested in the packaging process*: issue `make show` for a list of available build routines.

10 Configuration Files

> *Caution*: For the authoritative documentation, consult the manpage for `luaotfload.conf(5)`.

The runtime behavior of **Luaotfload** can be customized by means of a configuration file. At startup, it attempts to locate a file called `luaotfload.conf` or `luaotfloadrc` at a number of candidate locations:

- `./luaotfload.conf`
- `./luaotfloadrc`
- `$XDG_CONFIG_HOME/luaotfload/luaotfload.conf`
- `$XDG_CONFIG_HOME/luaotfload/luaotfload.rc`
- `/.luaotfloadrc`

> *Caution*: The configuration potentially modifies the final document. A project-local file belongs under version control along with the rest of the document. This is to ensure that everybody who builds the project also receives the same customizations as the author.

The syntax is fairly close to the format used by `git-config(1)` which in turn was derived from the popular .INI format: Lines of key-value pairs are grouped under different configuration "sections".[22] An example for customization via `luaotfload.conf` might look as below:

```
; Example luaotfload.conf containing a rudimentary configuration
[db]
  update-live = false
[run]
  color-callback = pre_linebreak_filter
  definer = info_patch
  log-level = 5
[default-features]
  global = mode=base
```

This specifies that for the given project, **Luaotfload** shall not attempt to automatically scan for fonts if it can't resolve a request. The font-based colorization will happen during LuaTEX's pre-linebreak filter. The fontloader will output verbose information about the fonts at definition time along with globally increased verbosity. Lastly, the fontloader defaults to the less expensive **base** mode like it does in ConTEXt.

11 Auxiliary Functions

With release version 2.2, **Luaotfload** received additional functions for package authors to call from outside (see the file `luaotfload-auxiliary.lua` for details). The purpose of this addition twofold. Firstly, **luaotfload** failed to provide a stable interface to internals in the past which resulted in an unmanageable situation of different packages abusing the raw

[22] The configuration parser in `luoatfload-parsers.lua` might be employed by other packages for similar purposes.

access to font objects by means of the patch_font callback. When the structure of the font object changed due to an update, all of these imploded and several packages had to be fixed while simultaneously providing fallbacks for earlier versions. Now the patching is done on the luaotfload side and can be adapted with future modifications to font objects without touching the packages that depend on it. Second, some the capabilities of the font loader and the names database are not immediately relevant in luaotfload itself but might nevertheless be of great value to package authors or end users.

Note that the current interface is not yet set in stone and the development team is open to suggestions for improvements or additions.

11.1 *Callback Functions*

The patch_font callback is inserted in the wrapper luaotfload provides for the font definition callback. At this place it allows manipulating the font object immediately after the font loader is done creating it. For a short demonstration of its usefulness, here is a snippet that writes an entire font object to the file `fontdump.lua`:

```
\input  luaotfload.sty
\directlua {
  local dumpfile    = "fontdump.lua"
  local dump_font   = function (tfmdata)
    local data = table.serialize(tfmdata)
    io.savedata(dumpfile, data)
  end
  luatexbase.add_to_callback(
    "luaotfload.patch_font",
    dump_font,
    "my_private_callbacks.dump_font"
  )
}
\font  \dumpme  = name:Iwona
\bye
```

Beware: this creates a Lua file of around 150,000 lines of code, taking up 3 MB of disk space. By inspecting the output you can get a first impression of how a font is structured in LuaTEX's memory, what elements it is composed of, and in what ways it can be rearranged.

11.1.1 Compatibility with Earlier Versions

As has been touched on in the preface to this section, the structure of the object as returned by the fontloader underwent rather drastic changes during different stages of its development, and not all packages that made use of font patching have kept up with every one of it. To ensure compatibility with these as well as older versions of some packages, luaotfload sets up copies of or references to data in the font table where it used to be located. For instance, important parameters like the requested point size, the units factor, and the font name have again been made accessible from the toplevel of the table even though they were migrated to different subtables in the meantime.

11.1.2 Patches

These are mostly concerned with establishing compatibility with XETEX.

- set_sscale_dimens
  Calculate \fontdimens 10 and 11 to emulate XeTeX.
- set_capheight
  Calculates \fontdimen 8 like XeTeX.
- patch_cambria_domh
  Correct some values of the font *Cambria Math*.

11.2 *Package Author's Interface*

As LuaTeX release 1.0 is nearing, the demand for a reliable interface for package authors increases.

11.2.1 Font Properties

Below functions mostly concern querying the different components of a font like for instance the glyphs it contains, or what font features are defined for which scripts.

- aux.font_has_glyph (id : int, index : int)
  Predicate that returns true if the font id has glyph index.
- aux.slot_of_name(id : int, name : string)
  Translates a name for a glyph in font id to the corresponding glyph slot which can be used e.g. as an argument to \char. The number is assigned by the luaotfload code and not related to the glyph index (GID) of the font as stored in the [index] field of the lua-file.
- aux.gid_of_name(id : int, name : string)
  New version 3.12
  Translates a name for a glyph in font id to the corresponding glyph index (GID) as stored in the [index] field.
- aux.name_of_slot(id : int, slot : int)
  The inverse of slot_of_name; note that this might be incomplete as multiple glyph names may map to the same codepoint, only one of which is returned by name_of_slot.
- aux.gid_of_name(id : int, name : string)
  Translates a Glyph name to the corresponding GID in font id. This corresponds to the value returned by \XeTeXglyphindex in XeTeX.
- aux.provides_script(id : int, script : string)
  Test if a font supports script.
- aux.provides_language(id : int, script : string, language : string)
  Test if a font defines language for a given script.
- aux.provides_feature(id : int, script : string, language : string, feature : string)
  Test if a font defines feature for language for a given script.
- aux.get_math_dimension(id : int, dimension : string)
  Get the dimension dimension of font id.
- aux.sprint_math_dimension(id : int, dimension : string)
  Same as get_math_dimension(), but output the value in scaled points at the TeX end.

11.2.2 Database

- aux.read_font_index (void)
  Read the index file from the appropriate location (usually the bytecode file `luaotfload-names.luc` somewhere in the `texmf-var` tree) and return the result as a table. The file is processed with each call so it is up to the user to store the result for later access.

- aux.font_index (void)
  Return a reference of the font names table used internally by luaotfload. The index will be read if it has not been loaded up to this point. Also a font scan that overwrites the current index file might be triggered. Since the return value points to the actual index, any modifications to the table might influence runtime behavior of luaotfload.

11.3 *Format Author's Interface*

Additionally some functions are provided which should only be needed for format authors trying to integrate luaotfload's color handling with the conventions of a specific format.

End users should never have to work with these.

11.3.1 Color setting

luaotfload's `color` feature can apply color in two different ways: By default it inserts `pdf_colorstack` whatsits containing the PDF code to change the color.

Alternatively a function can be provided which gets called on every glyph node to be colored. This function can then e.g. apply an attribute which gets interpreted later.

This can be enabled separately for colors and transparency by passing such callback functions:

- luaotfload.set_colorhandler((head : node, n : node, color : string) -> (head : node, n : node)
  Mark the node n in the list starting with head to be colored with color color.

  By default the color is represented by literal PDF code setting the color.

- luaotfload.set_transparenthandler((head : node, n : node, level : string) -> (head : node, n : node)
  Mark the node n in the list starting with head to be set transparently with transparency level level.

  By default the transparency level is represented by literal PDF code setting the transparency.

When these functions aren't used, then the color is set based on colorstack 0. By default a new colorstack is allocated for transparency, but alternatively an existsing colorstack for this prpose can be set:

- luaotfload.set_transparent_colorstack(stack : int)
  Use colorstack stack for setting transparency.

11.3.2 Color selection

Additionally the translation of the argument to `color` to an actual PDF color can be customized though three **exclusive** callbacks:

Since there is only a single `color` feature which sets both the color and the transparency, the first callback **luaotfload.split_color** gets called with a single string representing the feature value and is supposed to return two values representing the color and the transparency part.

The default implementation splits the string the feature value at a comma and strips outer spaces for both result values, except the HTML style 8 hexdigit RGBA values get split into the first 6 and the last two digits.

Afterwards the return values get passed to the **luaotfload.parse_color** respectively **luaotfload.parse_transparent** callbacks (except that **false** and **nil** skip the corresponding callback).

These callbacks should translate these strings components into valid PDF commands applyind the described color or transparency.

All error handling and reporting should be done in the callbacks.

12 Troubleshooting

12.1 *Database Generation*

If you encounter problems with some fonts, please first update to the latest version of this package before reporting a bug, as **luaotfload** is under active development and still a moving target. The development takes place on **github** at `https://github.com/lualatex/luaotfload` where there is an issue tracker for submitting bug reports, feature requests and the likes.

Bug reports are more likely to be addressed if they contain the output of

```
luaotfload-tool --diagnose=environment,files,permissions
```

Consult the man page for a description of these options.

Errors during database generation can be traced by increasing the verbosity level and redirecting log output to `stdout`:

```
luaotfload-tool -fuvvv --log=stdout
```

or to a file in `/tmp`:

```
luaotfload-tool -fuvvv --log=file
```

In the latter case, invoke the `tail(1)` utility on the file for live monitoring of the progress.

If database generation fails, the font last printed to the terminal or log file is likely to be the culprit. Please specify it when reporting a bug, and blacklist it for the time being (see above, page 25).

12.2 *Font Features*

A common problem is the lack of features for some **OpenType** fonts even when specified. This can be related to the fact that some fonts do not provide features for the `dflt`

script (see above on page 14), which is the default one in this package. If this happens, assigning a noth script when the font is defined should fix it. For example with `latn`:

```
\font  \test  = file:MyFont.otf:script=latn;+liga;
```

You can get a list of features that a font defines for scripts and languages by querying it in `luaotfload-tool`:

```
luaotfload-tool --find="Iwona" --inspect
```

12.3 *LuaTEX Programming*

Another strategy that helps avoiding problems is to not access raw LuaTEX internals directly. Some of them, even though they are dangerous to access, have not been overridden or disabled. Thus, whenever possible prefer the functions in the **aux** namespace over direct manipulation of font objects. For example, raw access to the **font.fonts** table like:

```
local somefont = font.fonts[2]
```

can render already defined fonts unusable. Instead, the function **font.getfont()** should be used because it has been replaced by a safe variant.

However, **font.getfont()** only covers fonts handled by the font loader, e. g. **OpenType** and **TrueType** fonts, but not TFM or OFM. Should you absolutely require access to all fonts known to LuaTEX, including the virtual and autogenerated ones, then you need to query both **font.getfont()** and **font.fonts**. In this case, best define you own accessor:

```
local unsafe_getfont = function (id)
    local tfmdata = font.getfont (id)
    if not tfmdata then
        tfmdata = font.fonts[id]
    end
    return tfmdata
end
--- use like getfont()
local somefont = unsafe_getfont (2)
```

13 License

luaotfload is licensed under the terms of the GNU General Public License version 2.0. Following the underlying fontloader code **luaotfload** recognizes only that exact version as its license. The „any later version” clause of the original license text as copyrighted by the Free Software Foundation *does not apply* to either **luaotfload** or the code imported from ConTEXt.

The complete text of the license is given as a separate file `COPYING` in the toplevel directory of the Luaotfload Git repository.
Distributions probably package it as `doc/luatex/luaotfload/COPYING` in the relevant `texmf` tree.

⟨*definition*⟩ ::= '\font', CSNAME, '=', ⟨*font request*⟩, [⟨*size*⟩] ;
⟨*size*⟩ ::= 'at', DIMENSION ;
⟨*font request*⟩ ::= '"', ⟨*unquoted font request*⟩ '"'
| '{', ⟨*unquoted font request*⟩ '}'
| ⟨*unquoted font request*⟩ ;
⟨*unquoted font request*⟩ ::= ⟨*specification*⟩, [':', ⟨*feature list*⟩]
| ⟨*path lookup*⟩, [[':'], ⟨*feature list*⟩] ;
⟨*specification*⟩ ::= ⟨*prefixed spec*⟩, [⟨*subfont no*⟩], { ⟨*modifier*⟩ }
| ⟨*anon lookup*⟩, { ⟨*modifier*⟩ } ;
⟨*prefixed spec*⟩ ::= 'combo:', ⟨*combo list*⟩
| 'file:', ⟨*file lookup*⟩
| 'name:', ⟨*name lookup*⟩ ;
⟨*combo list*⟩ ::= ⟨*combo def 1*⟩, { ';', ⟨*combo def*⟩ } ;
⟨*combo def 1*⟩ ::= ⟨*combo id*⟩, '->', ⟨*combo id*⟩ ;
⟨*combo def*⟩ ::= ⟨*combo id*⟩, '->', ⟨*combo id chars*⟩ ;
⟨*combo id*⟩ ::= ('(', { DIGIT }, ')' | { DIGIT }) ;
⟨*combo id chars*⟩ ::= ('(', { DIGIT }, ',', ⟨*combo chars*⟩, ')'
| { DIGIT }) ;
⟨*combo chars*⟩ ::= 'fallback'
| { ⟨*combo range*⟩, { '*', ⟨*combo range*⟩ } } ;
⟨*combo range*⟩ ::= ⟨*combo num*⟩, ['-', ⟨*combo num*⟩] ;
⟨*combo num*⟩ ::= '0x', { HEXDIGIT }
| 'U+', { DIGIT }
| { DIGIT } ;
⟨*file lookup*⟩ ::= { ⟨*name character*⟩ } ;
⟨*name lookup*⟩ ::= { ⟨*name character*⟩ } ;
⟨*anon lookup*⟩ ::= TFMNAME | ⟨*name lookup*⟩ ;
⟨*path lookup*⟩ ::= '[', { ⟨*path content*⟩ }, ']', [⟨*subfont no*⟩] ;
⟨*path content*⟩ ::= ⟨*path balanced*⟩
| '\', ALL_CHARACTERS
| ALL_CHARACTERS - ']'
⟨*path balanced*⟩ ::= '[', [⟨*path content*⟩], ']'
⟨*modifier*⟩ ::= '/', ('I' | 'B' | 'BI' | 'IB' | 'S=', { DIGIT }) ;
⟨*subfont no*⟩ ::= '(', { DIGIT }, ')' ;
⟨*feature list*⟩ ::= ⟨*feature expr*⟩, { ';', ⟨*feature expr*⟩ } ;
⟨*feature expr*⟩ ::= FEATURE_ID, '=', FEATURE_VALUE
| ⟨*feature switch*⟩, FEATURE_ID ;
⟨*feature switch*⟩ ::= '+' | '-' ;
⟨*name character*⟩ ::= ALL_CHARACTERS - ('(' | '/' | ':') ;

Figure 1: Font request syntax. Braces or double quotes around the *specification* rule will preserve whitespace in file names. In addition to the font style modifiers (*slash-notation*) given above, there are others that are recognized but will be silently ignored: `aat`, `icu`, and `gr`. The special terminals are: FEATURE_ID for a valid font feature name and FEATURE_VALUE for the corresponding value. TFMNAME is the name of a TFM file. DIGIT again refers to bytes 48–57, and ALL_CHARACTERS to all byte values. CSNAME and DIMENSION are the TEX concepts.

Luaotfload Libraries

luaotfload-filelist.lua
luaotfload-auxiliary.lua
luaotfload-colors.lua
luaotfload-configuration.lua
luaotfload-database.lua
luaotfload-features.lua
luaotfload-letterspace.lua
luaotfload-embolden.lua
luaotfload-notdef.lua
luaotfload-harf-define.lua
luaotfload-harf-plug.lua
luaotfload-loaders.lua
luaotfload-multiscript.lua
luaotfload-scripts.lua
luaotfload-szss.lua
luaotfload-fallback.lua
luaotfload-parsers.lua
luaotfload-resolvers.lua
luaotfload-unicode.lua
luaotfload-tounicode.lua
luaotfload-dvi.lua

luaotfload-database.lua → luaotfload-blacklist.cnf

luaotfload-database.lua → luaotfload-names.lua.gz
luaotfload-names.lua.gz
luaotfload-names.lua

main() → *Luaotfload Libraries*

Lualibs – Lua Libraries from Context

l-lua.lua
l-lpeg.lua
l-function.lua
l-string.lua
l-table.lua
l-io.lua
l-file.lua
l-boolean.lua
l-math.lua
l-unicode.lua
util-str.lua
util-fil.lua

luaotfload.lua → *Lualibs – Lua Libraries from Context*

luaotfload.lua → luaotfload-names.lua.gz
luaotfload-names.lua

--update

main()

luaotfload-log.lua

init_early() → luaotfload-log.lua

luaotfload-init.lua

init_early() → luaotfload-init.lua

unmerged → Merged libraries

Fontloader

fontloader-basics-gen.lua

fontloader-YY-MM-DD.lua

init_early() → fontloader-basics-gen.lua

init_main() → fontloader-YY-MM-DD.lua

merged → Merged libraries

Merged libraries

Font and Node Libraries from Context

data-con.lua
font-ini.lua
font-con.lua
font-cid.lua
font-map.lua
font-vfc.lua
font-otr.lua
font-cff.lua
font-ttf.lua
font-dsp.lua
font-oti.lua
font-ott.lua
font-otl.lua
font-oto.lua
font-otj.lua
font-oup.lua
font-ota.lua
font-ots.lua
font-otc.lua
font-osd.lua
font-ocl.lua
font-onr.lua
font-one.lua
font-afk.lua
font-lua.lua
font-def.lua
font-shp.lua
font-imp-tex.lua
font-imp-ligatures.lua
font-imp-italics.lua
font-imp-effects.lua

Font Loader (LuaTeX-Fonts)

luatex-basics-nod.lua
luatex-basics-chr.lua
luatex-fonts-mis.lua
luatex-fonts-enc.lua
luatex-fonts-tfm.lua
luatex-fonts-def.lua
luatex-fonts-ext.lua
luatex-fonts-lig.lua
luatex-fonts-gbn.lua

merges → fontloader-YY-MM-DD.lua

Standalone scripts

mkimport
mkglyphlist
mkcharacters
mkstatus
mktests

pulls → Merged libraries

pulls → *Font Loader (LuaTeX-Fonts)*

generates from glyphlist.txt → luaotfload-glyphlist.lua

generates from Context's char-def.lua → luaotfload-characters.lua

luatex-fonts-enc.lua → luaotfload-glyphlist.lua

luaotfload-auxiliary.lua → luaotfload-characters.lua

luaotfload-tool.lua

--diagnose → luaotfload-diagnostics.lua

luaotfload-diagnostics.lua

hash files → luaotfload-status.lua

version information → luaotfload-status.lua

generates from distribution files → luaotfload-status.lua

36

luaotfload.conf

Luaotfload configuration file

Date: 2022-10-03
Copyright: GPL v2.0
Version: 3.23
Manual section: 5
Manual group: text processing

SYNOPSIS

- **./luaotfload{.conf,rc}**
- **XDG_CONFIG_HOME/luaotfload/luaotfload{.conf,rc}**
- **~/.luaotfloadrc**

DESCRIPTION

The file `luaotfload.conf` contains configuration options for *Luaotfload*, a font loading and font management component for LuaTeX.

EXAMPLE

A small Luaotfload configuration file with few customizations could look as follows:

```
[db]
    formats = afm,ttf
    compress = false

[misc]
    termwidth = 60

[run]
    log-level = 6
```

This will make Luaotfload ignore all font files except for PostScript binary fonts with a matching AFM file, and Truetype fonts. Also, an uncompressed index file will be dumped which is going to be much larger than the default gzip'ed index. The terminal width is truncated to 60 characters which influences the verbose output during

indexing. Finally, the verbosity is increased greatly: each font file being processed will be printed to the stdout on a separate line, along with lots of other information.

To observe the difference in behavior, save above snippet to `./luaotfload.conf` and update the font index:

```
luaotfload-tool --update --force
```

The current configuration can be written to disk using **luaotfload-tool**:

```
luaotfload-tool --dumpconf > luaotfload.conf
```

The result can itself be used as a configuration file.

SYNTAX

The configuration file syntax follows the common INI format. For a more detailed description please refer to the section "CONFIGURATION FILE" of **git-config**(1). A brief list of rules is given below:

- Blank lines and lines starting with a semicolon (`;`) are ignored.
- A configuration file is partitioned into sections that are declared by specifying the section title in brackets on a separate line:

  ```
  [some-section]
  ... section content ...
  ```

- Sections consist of one or more variable assignments of the form `variable-name = value` E. g.:

  ```
  [foo]
      bar = baz
      quux = xyzzy
      ...
  ```

- Section and variable names may contain only uppercase and lowercase letters as well as dashes (`-`).

VARIABLES

Variables in belong into a configuration section and their values must be of a certain type. Some of them have further constraints. For example, the "color callback" must be a string of one of the values `post_linebreak_filter`, `pre_linebreak_filter`, or `pre_output_filter`, defined in the section *run* of the configuration file.

Currently, the configuration is organized into four sections:

db Options relating to the font index.

misc Options without a clearly defined category.

paths Path and file name settings.

run Options controlling runtime behavior of Luaotfload.

The list of valid variables, the sections they are part of and their type is given below. Types represent Lua types that the values must be convertible to; they are abbreviated as follows: `s` for the *string* type, `n` for *number*, `b` for *boolean*. A value of `nil` means the variable is unset.

Section db

variable	type	default
compress	b	true
designsize-dimen	b	bp
formats	s	"otf,ttf,ttc"
max-fonts	n	2^51
scan-local	b	false
skip-read	b	false
strip	b	true
update-live	b	true

The flag compress determines whether the font index (usually luaotfload-names.lua[.gz] will be stored in compressed forms. If unset it is equivalent of passing --no-compress to **luaotfload-tool**. Since the file is only created for convenience and has no effect on the runtime behavior of Luaotfload, the flag should remain set. Most editors come with zlib support anyways.

The setting designsize-dimen applies when looking up fonts from families with design sizes. In Opentype, these are specified as "decipoints" where one decipoint equals ten DTP style "big points". When indexing fonts these values are converted to sp. In order to treat the values as though they were specified in TeX points or Didot points, set designsize-dimen to pt or dd.

The list of formats must be a comma separated sequence of strings containing one or more of these elements:

- otf (OpenType format),
- ttf and ttc (TrueType format),
- afm (Adobe Font Metrics),

It corresponds loosely to the --formats option to **luaotfload-tool**. Invalid or duplicate members are ignored; if the list does not contain any useful identifiers, the default list "otf,ttf,ttc" will be used.

The variable max-fonts determines after processing how many font files the font scanner will terminate the search. This is useful for debugging issues with the font index and has the same effect as the option --max-fonts to **luaotfload-tools**.

The scan-local flag, if set, will incorporate the current working directory as a font search location. NB: This will potentially slow down document processing because a font index with local fonts will not be saved to disk, so these fonts will have to be re-indexed whenever the document is built.

The skip-read flag is only useful for debugging: It makes Luaotfload skip reading fonts. The font information for rebuilding the index is taken from the presently existing one.

Unsetting the strip flag prevents Luaotfload from removing data from the index that is only useful when processing font files. NB: this can increase the size of the index files significantly and has no effect on the runtime behavior.

If update-live is set, Luaotfload will reload the database if it cannot find a requested font. Those who prefer to update manually using **luaotfload-tool** should unset this flag. This option does not affect rebuilds due to version mismatch.

Section `default-features`

By default Luaotfload enables `node` mode and picks the default font features that are prescribed in the OpenType standard. This behavior may be overridden in the `default-features` section. Global defaults that will be applied for all scripts can be set via the `global` option, others by the script they are to be applied to. For example, a setting of

```
[default-features]
    global = mode=base,color=0000FF
    dflt   = smcp,onum
```

would force *base* mode, tint all fonts blue and activate small capitals and text figures globally. Features are specified as a comma separated list of variables or variable-value pairs. Variables without an explicit value are set to `true`.

Section `misc`

variable	type	default
statistics	b	`false`
termwidth	n	`nil`
version	s	<Luaotfload version>
keepnames	b	`true`

With `statistics` enabled, extra statistics will be collected during index creation and appended to the index file. It may then be queried at the Lua end or inspected by reading the file itself.

The value of `termwidth`, if set, overrides the value retrieved by querying the properties of the terminal in which Luatex runs. This is useful if the engine runs with `shell_escape` disabled and the actual terminal dimensions cannot be retrieved.

The value of `version` is derived from the version string hard-coded in the Luaotfload source. Override at your own risk.

The `keepnames` option decides if the ConTeXt fontloader should keep names it considers useless or if they should be discarded. This option only takes effect after font caches are regenererated.

Section `paths`

variable	type	default
cache-dir	s	`"fonts"`
names-dir	s	`"names"`
index-file	s	`"luaotfload-names.lua"`
lookups-file	s	`"luaotfload-lookup-cache.lua"`

The paths `cache-dir` and `names-dir` determine the subdirectory inside the Luaotfload subtree of the `luatex-cache` directory where the font cache and the font index will be stored, respectively.

Inside the index directory, the names of the index file and the font lookup cache will be derived from the respective values of `index-file` and `lookups-file`. This is the filename base for the bytecode compiled version as well as – for the index – the gzipped version.

Section `run`

variable	type	default
anon-sequence	s	`"tex,path,name"`
color-callback	s	`"post_linebreak_filter"`
definer	s	`"patch"`
log-level	n	`0`
resolver	s	`"cached"`
fontloader	s	`"default"`

Unqualified font lookups are treated with the flexible "anonymous" mechanism. This involves a chain of lookups applied successively until the first one yields a match. By default, the lookup will first search for TFM fonts using the Kpathsea library. If this wasn't successful, an attempt is made at interpreting the request as an absolute path (like the `[/path/to/font/foo.ttf]` syntax) or a file name (`file:foo.ttf`). Finally, the request is interpreted as a font name and retrieved from the index (`name:Foo Regular`). This behavior can be configured by specifying a list as the value to `anon-sequence`. Available items are `tex`, `path`, `name` – representing the lookups described above, respectively –, and `file` for searching a filename but not an absolute path. Also, `my` lookups are valid values but they should only be used from within TeX documents, because there is no means of customizing a `my` lookups on the command line.

The `color-callback` option determines the stage at which fonts that defined with a `color=xxyyzz` feature will be colorized. By default this happens in a `post_linebreak_filter` but alternatively the `pre_linebreak_filter` or `pre_output_filter` may be chosen, which is faster but might produce inconsistent output. The `pre_output_filter` used to be the default in the 1.x series of Luaotfload, whilst later versions up to and including 2.5 hooked into the `pre_linebreak_filter` which naturally didn't affect any glyphs inserting during hyphenation. Both are kept around as options to restore the previous behavior if necessary.

The `definer` allows for switching the `define_font` callback. Apart from the default `patch` one may also choose the `generic` one that comes with the vanilla fontloader. Beware that this might break tools like Fontspec that rely on the `patch_font` callback provided by Luaotfload to perform important corrections on font data.

The fontloader backend can be selected by setting the value of `fontloader`. The most important choices are `default`, which will load the dedicated Luaotfload fontloader, and `reference`, the upstream package as shipped with Luaotfload. Other than those, a file name accessible via kpathsea can be specified.

Alternatively, the individual files that constitute the fontloader can be loaded directly. While less efficient, this greatly aids debugging since error messages will reference the actual line numbers of the source files and explanatory comments are not stripped. Currently, three distinct loading strategies are available: `unpackaged` will load the batch that is part of Luaotfload. These contain the identical source code that

the reference fontloader has been compiled from. Another option, `context` will attempt to load the same files by their names in the Context format from the search path. Consequently this option allows to use the version of Context that comes with the TeX distribution. Distros tend to prefer the stable version ("current" in Context jargon) of those files so certain bugs encountered in the more bleeding edge Luaotfload can be avoided this way. A third option is to use `context` with a colon to specify a directory prefix where the *TEXMF* is located that the files should be loaded from, e. g. `context:~/context/tex/texmf-context`. This can be used when referencing another distribution like the Context minimals that is installed under a different path not indexed by kpathsea.

The value of `log-level` sets the default verbosity of messages printed by Luaotfload. Only messages defined with a verbosity of less than or equal to the supplied value will be output on the terminal. At a log level of five Luaotfload can be very noisy. Also, printing too many messages will slow down the interpreter due to line buffering being disabled (see **setbuf**(3)).

The `resolver` setting allows choosing the font name resolution function: With the default value `cached` Luaotfload saves the result of a successful font name request to a cache file to speed up subsequent lookups. The alternative, `normal` circumvents the cache and resolves every request individually. (Since to the restructuring of the font name index in Luaotfload 2.4 the performance difference between the cached and uncached lookups should be marginal.)

FILES

Luaotfload only processes the first configuration file it encounters at one of the search locations. The file name may be either `luaotfload.conf` or `luaotfloadrc`, except for the dotfile in the user's home directory which is expected at `~/.luaotfloadrc`.

Configuration files are located following a series of steps. The search terminates as soon as a suitable file is encountered. The sequence of locations that Luaotfload looks at is

i. The current working directory of the LuaTeX process.

ii. The subdirectory `luaotfload/` inside the XDG configuration tree, e. g. `/home/oenothea/config/luaotfload/`.

iii. The dotfile.

iv. The *TEXMF* (using kpathsea).

SEE ALSO

luaotfload-tool(1), **luatex**(1), **lua**(1)

- `texdoc luaotfload` to display the PDF manual for the *Luaotfload* package
- Luaotfload development https://github.com/latex3/luaotfload
- LuaLaTeX mailing list http://tug.org/pipermail/lualatex-dev/
- LuaTeX http://luatex.org/
- Luaotfload on CTAN http://ctan.org/pkg/luaotfload

REFERENCES

- The XDG base specification http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html.

AUTHORS

Luaotfload was developed by the LuaLaTeX dev team (https://github.com/lualatex/).
It is currently maintained by the LaTeX Project Team at https://github.com/latex3/luaotfload

This manual page was written by Philipp Gesang <phg@phi-gamma.net>.

luaotfload-tool

generate and query the Luaotfload font names database

Date: 2022-10-03
Copyright: GPL v2.0
Version: 3.23
Manual section: 1
Manual group: text processing

SYNOPSIS

luaotfload-tool [-bcDfFiIlLnpqRSuvVhw]

luaotfload-tool –update [–force] [–quiet] [–verbose] [–prefer-texmf] [–dry-run] [–formats=[+|-]EXTENSIONS] [–no-compress] [–no-strip] [–local] [–max-fonts=N]

luaotfload-tool –find=FONTNAME [–fuzzy] [–info] [–inspect] [–no-reload]

luaotfload-tool –flush-lookups
luaotfload-tool –cache=DIRECTIVE
luaotfload-tool –list=CRITERION[:VALUE] [–fields=F1,F2,…,Fn]
luaotfload-tool –bisect=DIRECTIVE
luaotfload-tool –help
luaotfload-tool –version
luaotfload-tool –show-blacklist
luaotfload-tool –diagnose=CHECK
luaotfload-tool –conf=FILE –dumpconf

DESCRIPTION

luaotfload-tool accesses the font names database that is required by the *Luaotfload* package. There are two general modes: **update** and **query**.

- **update**: update the database or rebuild it entirely;
- **query**: resolve a font name or display close matches.

OPTIONS

update mode

–update, -u Update the database; indexes new fonts.

–force, -f Force rebuilding of the database; re-indexes all fonts.

–local, -L Include font files in `$PWD`. This option will cause large parts of the database to be rebuilt. Thus it is quite inefficient. Additionally, if local font files are found, the database is prevented from being saved to disk, so the local fonts need to be parsed with every invocation of `luaotfload-tool`.

–no-reload, -n Suppress auto-updates to the database (e.g. when `--find` is passed an unknown name).

–no-compress, -c Do not filter the plain text version of the font index through gzip. Useful for debugging if your editor is built without zlib.

–prefer-texmf, -p Organize the file name database in a way so that it prefer fonts in the *TEXMF* tree over system fonts if they are installed in both.

–formats=EXTENSIONS Extensions of the font files to index. Where *EXTENSIONS* is a comma-separated list of supported file extensions (otf, ttf, ttc). If the list is prefixed with a + sign, the given list is added to the currently active one; - subtracts. Default: *otf,ttf,ttc*. Examples:

1) `--formats=-ttc,ttf` would skip TrueType fonts and font collections;
2) `--formats=otf` would scan only OpenType files;
3) `--formats=+afm` includes binary Postscript files accompanied by an AFM file.

query mode

–find=NAME Resolve a font name; this looks up <name> in the database and prints the file name it is mapped to. `--find` also understands request syntax, i.e. `--find=file:foo.otf` checks whether `foo.otf` is indexed.

–fuzzy, -F Show approximate matches to the file name if the lookup was unsuccessful (requires `--find`).

–info, -i Display basic information to a resolved font file (requires `--find`).

–inspect, -I Display detailed information by loading the font and analyzing the font table; very slow! For the meaning of the returned fields see the LuaTeX documentation. (requires `--find`).

–list=CRITERION Show entries, where *CRITERION* is one of the following:

1) the character *, selecting all entries;
2) a field of a database entry, for instance *version* or *format**, according to which the output will be sorted. Information in an unstripped database (see the option `--no-strip` above) is nested: Subfields of a record can be addressed using the `->` separator, e. g. `file->location`, `style->units_per_em`, or `names->sanitized->english->prefmodifiers`. NB: shell syntax requires that arguments containing `->` be properly quoted!
3) an expression of the form `field:value` to limit the output to entries whose `field` matches `value`. The `value` can contain * to match an arbitrary number of characters.

For example, in order to output file names and corresponding versions, sorted by the font format:

```
./luaotfload-tool.lua --list="format" --fields="file->base,version"
```

This prints:

```
otf latinmodern-math.otf  Version 1.958
otf lmromancaps10-oblique.otf 2.004
otf lmmono8-regular.otf 2.004
otf lmmonoproplt10-bold.otf 2.004
otf lmsans10-oblique.otf  2.004
otf lmromanslant8-regular.otf 2.004
otf lmroman12-italic.otf  2.004
otf lmsansdemicond10-oblique.otf  2.004
...
```

–fields=FIELDS Comma-separated list of fields that should be printed. Information in an unstripped database (see the option `--no-strip` above) is nested: Subfields of a record can be addressed using the `->` separator, e. g. `file->location`, `style->units_per_em`, or `names->sanitized->english->subfamily`. The default is plainname,version*. (Only meaningful with `--list`.)

font and lookup caches

–flush-lookups Clear font name lookup cache (experimental).

–cache=DIRECTIVE Cache control, where *DIRECTIVE* is one of the following:

1) `purge` -> delete Lua files from cache;
2) `erase` -> delete Lua and Luc files from cache;
3) `show` -> print stats.

debugging methods

–show-blacklist, -b Show blacklisted files (not directories).

–dry-run, -D Don't load fonts when updating the database; scan directories only. (For debugging file system related issues.)

–no-strip Do not strip redundant information after building the database. Warning: this will inflate the index to about two to three times the normal size.

–max-fonts=N Process at most *N* font files, including fonts already indexed in the count.

–bisect=DIRECTIVE Bisection of the font database. This mode is intended as assistance in debugging the Luatex engine, especially when tracking memleaks or buggy fonts.

DIRECTIVE can be one of the following:

1) `run` -> Make `luaotfload-tool` respect the bisection progress when running. Combined with `--update` and possibly `--force` this will only process the files from the start up until the pivot and ignore the rest.
2) `start` -> Start bisection: create a bisection state file and initialize the low, high, and pivot indices.
3) `stop` -> Terminate the current bisection session by deleting the state file.
4) `good` | `bad` -> Mark the section processed last as "good" or "bad", respectively. The next bisection step will continue with the bad section.
5) `status` -> Print status information about the current bisection session. Hint: Use with higher verbosity settings for more output.

A bisection session is initiated by issuing the `start` directive. This sets the pivot to the middle of the list of available font files. Now run *luaotfload-tool* with the `--update` flag set as well as `--bisect=run`: only the fonts up to the pivot will be considered. If that task exhibited the issue you are tracking, then tell Luaotfload using `--bisect=bad`. The next step of `--bisect=run` will continue bisection with the part of the files below the pivot. Likewise, issue `--bisect=good` in order to continue with the fonts above the pivot, assuming the tested part of the list did not trigger the bug.

Once the culprit font is tracked down, `good` or `bad` will have no effect anymore. `run` will always end up processing the single font file that was left. Use `--bisect=stop` to clear the bisection state.

miscellaneous

–verbose=N, -v Set verbosity level to *n* or the number of repetitions of `-v`.

–quiet No verbose output (log level set to zero).

–log=CHANNEL Redirect log output (for database troubleshooting), where *CHANNEL* can be

1) `stdout` -> all output will be dumped to the terminal (default); or
2) `file` -> write to a file to the temporary directory (the name will be chosen automatically.

–version, -V Show version numbers of components as well as some basic information and exit.

–help, -h Show help message and exit.

–diagnose=CHECK Run the diagnostic procedure *CHECK*. Available procedures are:

1) `files` -> check *Luaotfload* files for modifications;
2) `permissions` -> check permissions of cache directories and files;
3) **`environment` -> print relevant** environment and kpse variables;
4) `repository` -> check the git repository for new releases,

5) `index` -> check database, display information about it.

Procedures can be chained by concatenating with commas, e.g. `--diagnose=files,permissions`. Specify `thorough` to run all checks.

–conf=FILE Read the configuration from *FILE*. See **luaotfload.conf**(%) for documentation concerning the format and available options.

–dumpconf Print the currently active configuration; the output can be saved to a file and used for bootstrapping a custom configuration files.

–aliases Dump the font name database as a kpathsea aliases file. This option is experimental and might go away.

FILES

The font name database is usually located in the directory `texmf-var/luatex-cache/generic/names/` (`$TEXMFCACHE` as set in `texmf.cnf`) of your *TeX Live* distribution as a zlib-compressed file `luaotfload-names.lua.gz`. The experimental lookup cache will be created as `luaotfload-lookup-cache.lua` in the same directory. These Lua tables are not used directly by Luaotfload, though. Instead, they are compiled to Lua bytecode which is written to corresponding files with the extension `.luc` in the same directory. When modifying the files by hand keep in mind that only if the bytecode files are missing will Luaotfload use the plain version instead. Both kinds of files are safe to delete, at the cost of regenerating them with the next run of *LuaTeX*.

SEE ALSO

luaotfload.conf(5), **luatex**(1), **lua**(1)

- `texdoc luaotfload` to display the manual for the *Luaotfload* package
- Luaotfload development https://github.com/latex3/luaotfload
- LuaLaTeX mailing list http://tug.org/pipermail/lualatex-dev/
- LuaTeX http://luatex.org/
- ConTeXt http://wiki.contextgarden.net
- Luaotfload on CTAN http://ctan.org/pkg/luaotfload

BUGS

Tons, probably.

AUTHORS

Luaotfload was developed by the LuaLaTeX dev team (https://github.com/lualatex/). It is currently maintained by the LaTeX Project Team at https://github.com/latex3/luaotfload The fontloader code is provided by Hans Hagen of Pragma ADE, Hasselt NL (http://pragma-ade.com/).

This manual page was written by Philipp Gesang <phg@phi-gamma.net>.