The luaotfload package

Elie Roux · Khaled Hosny · Philipp Gesang
Home: https://github.com/lualatex/luaotfload
Support: lualatex-dev@tug.org

2016/04/21 v2.7

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 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.

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.

[1] Unfortunately, luaotfload doesn't support many Indic scripts right now. Assistance in implementing the prerequisites is greatly appreciated.

2 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.

3 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.

3.1 *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 locations and in your `texmf` to be accessible by font name, **luaotfload** must first collect

[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 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.

the metadata included in the files. Please refer to section 6 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`.

3.2 *Compatibility Layer*

In addition to the regular prefixed requests, **luaotfload** accepts loading fonts the XeTeX way. There are again two modes: bracketed and unbracketed. A bracketed request looks as follows.

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

Inside the square brackets, every character except for a closing bracket is permitted, allowing for specifying paths to a font file. Naturally, path-less file names are equally valid and processed the same way as an ordinary `file:` lookup.

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

Unbracketed (or, for lack of a better word: *anonymous*) font requests resemble the conventional TeX syntax. 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 `name:` lookup, which itself will fall back to a `file:` lookup if no database entry matches ⟨*font name*⟩.

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.

3.3 *Examples*

3.3.1 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:

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

```
\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
```

3.3.2 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 }
{\pagellabold         foo bar baz\endgraf }
{\pagellabolditalic   foo bar baz\endgraf }
...
```

3.3.3 Modifiers

If the entire *Iwona* family[5] 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
```

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

```
\font  \iwonabolditalic  = "Iwona BoldItalic"   at 20pt
```

4 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[6] 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
```

4.1 *Basic font features*

- **mode**
  luaotfload has two OpenType processing *modes*: base and node.

  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.

- **script**
  An OpenType script tag;[7] the default value is `dlft`. Some fonts, including very

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

[7] 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.

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,[8] 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"
```

- **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. Ligatures are split into their component glyphs unless explicitly ignored (see below).

  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"
```

  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.

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

- **protrusion & expansion**
  These keys control microtypographic features of the font, namely *character protrusion* and *font expansion*. Their arguments are names of Lua tables that contain values for the respective features.[9] For both, only the set **default** is predefined.

  For example, to define a font with the default protrusion vector applied[10]:

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

4.2 *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[11]:

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

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

5 Combining fonts

Version 2.7 and later support combining 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.

[9] 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`.

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

[11] 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.

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

5.1 *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 поместья, 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.

5.2 *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.

6 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.[12]

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.

6.1 *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`.[13] 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
```

[12] 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.

[13] 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.

6.2 *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`

6.3 *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.[14]

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

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

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

`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`).[15]

6.4 *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,[16] 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
```

7 The Fontloader

7.1 *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,

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

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

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.

7.2 *Contents and Dependencies*

Below is a commented list of the files distributed with **luaotfload** in one way or the other. See figure 2 on page 22 for a graphical representation of the dependencies. Through the script `mkimport` a ConTEXt library is invoked to create the **luaotfload** fontloader as a merged (amalgamated) source file.[17] 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` 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.[18]

- `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`
- `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.

[17] 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.

[18] 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.

- luatex-basics-nod.lua
- luatex-basics-chr.lua
- luatex-fonts-enc.lua
- luatex-fonts-syn.lua
- luatex-fonts-ext.lua

- Code related to *font handling and node processing*, taken directly from ConTEXt.

  - data-con.lua
  - font-ini.lua
  - font-con.lua
  - font-cid.lua
  - font-map.lua
  - font-tfm.lua
  - font-afm.lua
  - font-afk.lua
  - font-oti.lua
  - font-otr.lua
  - font-cff.lua
  - font-ttf.lua
  - font-dsp.lua
  - font-oup.lua
  - font-otl.lua
  - font-oto.lua
  - font-otj.lua
  - font-ota.lua
  - font-ots.lua
  - font-osd.lua
  - font-lua.lua
  - font-def.lua
  - font-xtx.lua
  - font-gbn.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 8 and 7.3 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`.[19] 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

[19] 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-parsers.lua` – various LPEG-based parsers.
- `luaotfload-database.lua` – font names database.
- `luaotfload-resolvers.lua` – file name resolvers.
- `luaotfload-colors.lua` – color handling.
- `luaotfload-auxiliary.lua` – access to internal functionality for package authors (proposals for additions welcome).
- `luaotfload-letterspace.lua` – font-based letterspacing.

7.3 *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 8) provides the option `fontloader`. Its value can be one of the identifiers `default` or `reference` (see above, section 7.2) 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.[20] 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.

8 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".[21] 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
```

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

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

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.

9 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 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.

9.1 *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.

9.1.1 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.

9.1.2 Patches

These are mostly concerned with establishing compatibility with XETEX.

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

9.2 *Package Author's Interface*

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

9.2.1 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(name : string)*
  Translates an Adobe Glyph name to the corresponding glyph slot.
- *aux.name_of_slot(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.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.

9.2.2 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**.

10 Troubleshooting

10.1 *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 11).

10.2 *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 6), 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
```

10.3 *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)
```

11 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*⟩ ::= { ALL_CHARACTERS - '`]`' } ;

⟨*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*⟩ ::= '`+`' | '`-`' ;

Figure 2: Schematic of the files in Luaotfload