

# The luaotfload package

Elie Roux · Khaled Hosny · Philipp Gesang  
Home: <https://github.com/lualatex/luaotfload>  
Support: [lualatex-dev@tug.org](mailto:lualatex-dev@tug.org)

2013/07/23 v2.3b

## Abstract

This package is an adaptation of the ConT<sub>E</sub>Xt font loading system. It allows for loading OpenType fonts with an extended syntax and adds support for a variety of font features.

## Contents

I	Package Description	1
1	Introduction	1
2	Thanks	2
3	Loading Fonts	2
3.1	Prefix – the luaotfload Way	2
3.2	Compatibility Layer	3
3.3	Examples	3
3.3.1	Loading by File Name	3
3.3.2	Loading by Font Name	4
3.3.3	Modifiers	4
4	Font features	5
5	Font names database	7
5.1	luaotfload-tool / mkluatexfontdb.lua	7
5.2	Search Paths	8
5.3	Querying from Outside	8
5.4	Blacklisting Fonts	9
6	Files from ConT <sub>E</sub> Xt and LuaT <sub>E</sub> X-Fonts	10
7	Auxiliary Functions	11
7.1	Callback Functions	12
7.1.1	Compatibility with Earlier Versions	12
7.1.2	Patches	13
7.2	Package Author's Interface	13

7.2.1	Font Properties . . . . .	13
7.2.2	Database . . . . .	14
8	Troubleshooting . . . . .	14
II	Implementation . . . . .	14
9	luaotfload.lua . . . . .	14
9.1	Module loading . . . . .	16
9.2	Preparing the Font Loader . . . . .	17
9.3	Callbacks . . . . .	20
9.4	ConTeXt override . . . . .	24
10	luaotfload.sty . . . . .	25
11	The GNU GPL License v2 . . . . .	28

## I Package Description

### 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<sup>1</sup> 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  $\XeTeX$ , Lua $\TeX$  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 Lua $\TeX$  with functionality necessary for handling OpenType fonts. Additionally, it provides means for accessing fonts known to the operating system conveniently by indexing the metadata.

### 2 THANKS

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

---

<sup>1</sup>Unfortunately, luaotfload doesn't support many Indic scripts right now. Assistance in implementing the prerequisites is greatly appreciated.

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  $\TeX$  a breeze due to the extra effort he invested into isolating it from the rest of Con $\TeX$ t, 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:`.<sup>2</sup> 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.<sup>3</sup> In order for fonts installed both in system 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 5 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  $\TeX$  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`.

<sup>2</sup>The development version 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](#).

<sup>3</sup>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.

### 3.2 Compatibility Layer

In addition to the regular prefixed requests, `luaotfload` accepts loading fonts the  $\text{\XeTeX}$  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  $\text{\TeX}$  syntax. However, they have a broader spectrum of possible interpretations: before anything else, `luaotfload` attempts to load a traditional  $\text{\TeX}$  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  $\text{\XeTeX}$ .

```
\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  $\text{\XeTeX}$  engine and have no meaning in  $\text{\LuaTeX}$ .

### 3.3 Examples

#### 3.3.1 Loading by File Name

For example, conventional *TYPE1* 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*<sup>4</sup> in its condensed variant can be loaded as follows:

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

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

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

---

<sup>4</sup><http://jmn.pl/antykwa-poltawskiego/>, also available in  $\text{\TeX}$  Live.

### 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<sup>5</sup> 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
```

---

<sup>5</sup><http://jmn.pl/kurier-i-iwona/>, also in T<sub>E</sub>X Live.

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 (X<sub>Y</sub>TeX style), they must precede *<font features>*.

The element *<font features>*<sup>6</sup> is a semicolon-separated list of feature tags<sup>6</sup> 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
```

Other font options include:

### mode

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

base mode works by mapping OpenType features to traditional T<sub>E</sub>X 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 T<sub>E</sub>X'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,<sup>7</sup> 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.

<sup>6</sup>Cf. <http://www.microsoft.com/typography/otspec/featurelist.htm>.

<sup>7</sup>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.

### language

An OpenType language system identifier,<sup>8</sup> defaulting to `df1t`.

### featurefile

A comma-separated list of feature files to be applied to the font. Feature files contain a textual representation of OpenType tables and extend the features of a font on fly. After they are applied to a font, features defined in a feature file can be enabled or disabled just like any other font feature. The syntax is documented in Adobe's OpenType Feature File Specification.<sup>9</sup>

For a demonstration of how to set a `tkrn` feature consult the file `tkrn.fea` that is part of `luaotfload`. It can be read and applied as follows:

```
\font\test=Latin Modern Roman:featurefile=tkrn.fea;+tkrn
```

### 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
```

### 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.<sup>10</sup> For both, only the set `default` is predefined.

For example, to enable default protrusion<sup>11</sup>:

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

**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 (2013) there are three of them:

---

<sup>8</sup>Cf. <http://www.microsoft.com/typography/otspec/languagetags.htm>.

<sup>9</sup>Cf. [http://www.adobe.com/devnet/opentype/afdko/topic\\_feature\\_file\\_syntax.html](http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html).

<sup>10</sup>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`.

<sup>11</sup>You also need to set `pdfprotrudechars=2` and `pdfadjustspacing=2` to activate protrusion and expansion, respectively. See the [pdf \$\TeX\$  manual](#) for details.

**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 T<sub>E</sub>X ligatures:

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

<sup>12</sup>

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

## 5 FONT NAMES DATABASE

As mentioned above, `luaotfload` keeps track of which fonts are available to LuaT<sub>E</sub>X 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.<sup>13</sup>

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.

### 5.1 `luaotfload-tool /mkluatexfontdb.lua`<sup>14</sup>

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`.<sup>15</sup> Invoked with the argument `--update` it will perform a database update, scanning for fonts not indexed.

<sup>12</sup>These contain the feature set `trep` of earlier versions of `luaotfload`.

Note to X<sub>Y</sub>T<sub>E</sub>X users: this is the equivalent of the assignment `mapping=text-tex` using X<sub>Y</sub>T<sub>E</sub>X’s input remapping feature.

<sup>13</sup>The tool `otfinfo` (comes with T<sub>E</sub>X Live), when invoked on a font file with the `-i` option, lists the variety of name fields defined for it.

<sup>14</sup>The script may be named just `mkluatexfontdb` in your distribution.

<sup>15</sup>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.

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

---

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

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

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

For sake of backwards compatibility, `luaotfload-tool` may be renamed or sym-linked to `mkluatexfontdb`. Whenever it is run under this name, it will update the database first, mimicking the behavior of earlier versions of `luaotfload`.

## 5.2 *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 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.

## 5.3 *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 Lua $\TeX$  reference manual.<sup>16</sup>

`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`).<sup>17</sup>

#### 5.4 Blacklisting Fonts

Some fonts are problematic in general, or just in Lua $\TeX$ . 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,<sup>18</sup> 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:

```
% example otf-blacklist.cnf
/Library/Fonts/GillSans.ttc % Luaotfload ignores this font.
-/Library/Fonts/Optima.ttc % This one is usable again, even if
                             % blacklisted somewhere else.
```

## 6 FILES FROM CON $\TeX$ T AND LUA $\TeX$ -FONTS

`luaotfload` relies on code originally written by Hans Hagen<sup>19</sup> for and tested with Con $\TeX$ t. It integrates the font loader as distributed in the Lua $\TeX$ -Fonts package. The origi-

<sup>16</sup>In  $\TeX$  Live: `texmf-dist/doc/luatex/base/luatexref-t.pdf`.

<sup>17</sup>Or see `luaotfload-tool.rst` in the source directory.

<sup>18</sup>You may have to run `mktexlsr` if you created a new file in your `texmf` tree.

<sup>19</sup>The creator of the Con $\TeX$ t format.

nal Lua source files have been combined using the `mtx-package` script into a single, self-contained blob. In this form the font loader has no further dependencies<sup>20</sup> and requires only minor adaptations to integrate into `luaotfload`. The guiding principle is to let ConTeXt/LuaTeX-Fonts take care of the implementation, and update the imported code from time to time. 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.

Below is a commented list of the files distributed with `luaotfload` in one way or the other. See figure 2 on page 27 for a graphical representation of the dependencies. From LuaTeX-Fonts, only the file `luatex-fonts-merged.lua` has been imported as `luaotfload-merged.lua`. It is generated by `mtx-package`, a Lua source code merging tool developed by Hans Hagen.<sup>21</sup> It houses several Lua files that can be classed in three categories.

- *Lua utility libraries*, a subset of what is provided by the `lualibs` package.

– <code>l-lua.lua</code>	– <code>l-io.lua</code>
– <code>l-lpeg.lua</code>	– <code>l-file.lua</code>
– <code>l-function.lua</code>	– <code>l-boolean.lua</code>
– <code>l-string.lua</code>	– <code>l-math.lua</code>
– <code>l-table.lua</code>	– <code>util-str.lua</code>

- The *font loader* itself. These files have been written for LuaTeX-Fonts and they are distributed along with `luaotfload`.

– <code>luatex-basics-gen.lua</code>	– <code>luatex-fonts-chr.lua</code>
– <code>luatex-basics-nod.lua</code>	– <code>luatex-fonts-lua.lua</code>
– <code>luatex-fonts-enc.lua</code>	– <code>luatex-fonts-def.lua</code>
– <code>luatex-fonts-syn.lua</code>	– <code>luatex-fonts-ext.lua</code>
– <code>luatex-fonts-tfm.lua</code>	– <code>luatex-fonts-cbk.lua</code>

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

– <code>data-con.lua</code>	– <code>font-map.lua</code>
– <code>font-ini.lua</code>	– <code>font-oti.lua</code>
– <code>font-con.lua</code>	– <code>font-otf.lua</code>
– <code>font-cid.lua</code>	– <code>font-otb.lua</code>

<sup>20</sup>It covers, however, to some extent the functionality of the `lualibs` package.

<sup>21</sup>`mtx-package` is part of ConTeXt and requires `mtxrun`. 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.

- node-inj.lua	- font-def.lua
- font-ota.lua	
- font-otn.lua	- font-otp.lua

Note that if `luaotfload` cannot locate the merged file, it will load the individual Lua libraries instead. Their names remain the same as in ConT<sub>E</sub>Xt (without the `otfl`-prefix) since we imported the relevant section of `luatex-fonts.lua` unmodified into `luaotfload.lua`. Thus if you prefer running bleeding edge code from the ConT<sub>E</sub>Xt beta, all you have to do is remove `luaotfload-merged.lua` from the search path.

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`.<sup>22</sup> There is a make target `glyphs` that will create a fresh glyph list so we don't need to import it from ConT<sub>E</sub>Xt any longer.

In addition to these, `luaotfload` requires a number of files not contained in the merge. Some of these have no equivalent in LuaT<sub>E</sub>X-Fonts or ConT<sub>E</sub>Xt, some were taken unmodified from the latter.

- `luaotfload-features.lua` - font feature handling; incorporates some of the code from `font-otc` from ConT<sub>E</sub>Xt;
- `luaotfload-override.lua` - overrides the ConT<sub>E</sub>Xt logging functionality.
- `luaotfload-loaders.lua` - registers the OpenType font reader as handler for Postscript fonts (*PFA*, *PFB*).
- `luaotfload-database.lua` - font names database.
- `luaotfload-colors.lua` - color handling.
- `luaotfload-auxiliary.lua` - access to internal functionality for package authors (proposals for additions welcome).

## 7 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

<sup>22</sup>See `luaotfload-font-enc.lua`. The hard-coded file name is why we have to replace the procedure that loads the file in `luaotfload-override.lua`.

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.

## 7.1 *Callback Functions*

The *patch\_font* callback is inserted in the wrapper `luaotfload` provides for the font definition callback (see below, page 24). 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 Lua $\TeX$ 's memory, what elements it is composed of, and in what ways it can be rearranged.

### 7.1.1 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.

### 7.1.2 Patches

These are mostly concerned with establishing compatibility with X $\LaTeX$ .

- *set\_sscale\_dimens*  
Calculate `\fontdimens` 10 and 11 to emulate X<sub>Y</sub>TeX.
- *set\_capheight*  
Calculates `\fontdimen` 8 like X<sub>Y</sub>TeX.
- *patch\_cambria\_domh*  
Correct some values of the font *Cambria Math*.

## 7.2 Package Author's Interface

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

### 7.2.1 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.

### 7.2.2 Database

- *aux.scan\_external\_dir*(*dir* : string)  
Include fonts in directory *dir* in font lookups without adding them to the database.

## 8 TROUBLESHOOTING

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 requests and the likes.

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

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

If this fails, the font last printed to the terminal is likely to be the culprit. Please specify it when reporting a bug, and blacklist it for the time being (see above, page 9).

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 `df1t` 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;
```

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.

## II Implementation

### 9 luaotfload.lua

This file initializes the system and loads the font loader. To minimize potential conflicts between other packages and the code imported from ConTeXt, several precautions are in order. Some of the functionality that the font loader expects to be present, like raw access to callbacks, are assumed to have been disabled by `luatexbase` when this file is processed. In some cases it is possible to trick it by putting dummies into place and restoring the behavior from `luatexbase` after initialization. Other cases such as attribute allocation require that we hook the functionality from `luatexbase` into locations where they normally wouldn't be.

Anyways we can import the code base without modifications, which is due mostly to the extra effort by Hans Hagen to make Lua<sub>T</sub><sub>E</sub>X-Fonts self-contained and encapsulate it, and especially due to his willingness to incorporate our suggestions.

```

1 luaotfload                = luaotfload or {}
2 local luaotfload          = luaotfload
3
4 config                     = config or { }
5 config.luaotfload          = config.luaotfload or { }
6 -----.luaotfload.resolver = config.luaotfload.resolver      or "normal"
7 config.luaotfload.resolver = config.luaotfload.resolver      or "cached"
8 config.luaotfload.definer  = config.luaotfload.definer       or "patch"
9 config.luaotfload.compatibility = config.luaotfload.compatibility or false
10 config.luaotfload.loglevel = config.luaotfload.loglevel      or 1
11 config.luaotfload.color_callback = config.luaotfload.color_callback or "pre_line-break_filter"
12 config.luaotfload.prioritize = config.luaotfload.prioritize  or "sys"
13 config.luaotfload.names_dir  = config.luaotfload.names_dir   or "names"
14 config.luaotfload.cache_dir  = config.luaotfload.cache_dir   or "fonts"
15 config.luaotfload.index_file = config.luaotfload.index_file   or "luaotfload-names.lua"
16
17 luaotfload.module = {
18   name      = "luaotfload",
19   version   = 2.3002,
20   date      = "2013/07/23",
21   description = "OpenType layout system.",
22   author     = "Elie Roux & Hans Hagen",
23   copyright  = "Elie Roux",
24   license    = "GPL v2.0"
25 }
26
27 local luatexbase          = luatexbase
28
29 local setmetatable        = setmetatable
30 local type, next          = type, next
31
32 local kpsefind_file       = kpse.find_file
33 local lfsisfile           = lfs.isfile
34
35 local add_to_callback, create_callback =
36   luatexbase.add_to_callback, luatexbase.create_callback
37 local reset_callback, call_callback =
38   luatexbase.reset_callback, luatexbase.call_callback
39
40 local dummy_function = function () end
41
42 local error, warning, info, log =
43   luatexbase.provides_module(luaotfload.module)

```



```

44
45 luaotfload.error      = error
46 luaotfload.warning    = warning
47 luaotfload.info       = info
48 luaotfload.log        = log
49

```

We set the minimum version requirement for Lua<sub>T</sub><sub>E</sub>X to v0.76, because the font loader requires recent features like direct attribute indexing and *node.end\_of\_math()* that aren't available in earlier versions.<sup>23</sup>

```

50
51 local luatex_version = 76
52
53 if tex.luatexversion < luatex_version then
54     warning("LuaTeX v%.2f is old, v%.2f is recommended.",
55           tex.luatexversion/100,
56           luatex_version /100)
57     --- we install a fallback for older versions as a safety
58     if not node.end_of_math then
59         local math_t      = node.id"math"
60         local traverse_nodes = node.traverse_id
61         node.end_of_math = function (n)
62             for n in traverse_nodes(math_t, n.next) do
63                 return n
64             end
65         end
66     end
67 end
68

```

## 9.1 Module loading

We load the files imported from Con<sub>T</sub><sub>E</sub>Xt with this function. It automatically prepends the prefix `luaotfload-` to its argument, so we can refer to the files with their actual Con<sub>T</sub><sub>E</sub>Xt name.

```

69
70 local fl_prefix = "luaotfload" -- "luatex" for luatex-plain
71 local loadmodule = function (name)
72     require(fl_prefix .. "-" .. name)
73 end
74

```

Before <sub>T</sub><sub>E</sub>XLive 2013 version, Lua<sub>T</sub><sub>E</sub>X had a bug that made ofm fonts fail when called with their extension. There was a side-effect making ofm totally unloadable when `luaotfload` was present. The following lines are a patch for this bug. The utility of these lines is

---

<sup>23</sup>See Taco's announcement of v0.76: <http://comments.gmane.org/gmane.comp.tex.luatex.user/4042> and this commit by Hans that introduced those features. <http://repo.or.cz/w/context.git/commitdiff/a51f6cf6ee087046a2ae5927ed4edff0a1acec1b>.

questionable as they are not necessary since  $\TeX$ Live 2013. They should be removed in the next version.

```

75 local Cs, P, lpegmatch = lpeg.Cs, lpeg.P, lpeg.match
76
77 local p_dot, p_slash = P".", P"/"
78 local p_suffix      = (p_dot * (1 - p_dot - p_slash)^1 * P(-1)) / ""
79 local p_removesuffix = Cs((p_suffix + 1)^1)
80
81 local find_vf_file = function (name)
82     local fullname = kpsefind_file(name, "ovf")
83     if not fullname then
84         --fullname = kpsefind_file(file.removesuffix(name), "ovf")
85         fullname = kpsefind_file(lpegmatch(p_removesuffix, name), "ovf")
86     end
87     if fullname then
88         log("loading virtual font file %s.", fullname)
89     end
90     return fullname
91 end
92

```

## 9.2 Preparing the Font Loader

We treat the fontloader as a black box so behavior is consistent between formats. We do no longer run the intermediate wrapper file `luaotfload-fonts.lua` which we used to import from [Lua \$\TeX\$ -Plain](#). Rather, we load the fontloader code directly in the same fashion as `luatex-fonts`. How this is executed depends on the presence on the *merged font loader code*. In `luaotfload` this is contained in the file `luaotfload-merged.lua`. If this file cannot be found, the original libraries from Con $\TeX$ t of which the merged code was composed are loaded instead. The imported font loader will call `callback.register` once while reading `font-def.lua`. This is unavoidable unless we modify the imported files, but harmless if we make it call a dummy instead. However, this problem might vanish if we decide to do the merging ourselves, like the `lua-libs` package does. With this step we would obtain the freedom to load our own overrides in the process right where they are needed, at the cost of losing encapsulation. The decision on how to progress is currently on indefinite hold.

```

93
94 local starttime = os.gettimeofday()
95
96 local trapped_register = callback.register
97 callback.register      = dummy_function
98

```

By default, the fontloader requires a number of *private attributes* for internal use. These must be kept consistent with the attribute handling methods as provided by `luatexbase`. Our strategy is to override the function that allocates new attributes before we initialize the font loader, making it a wrapper around `luatexbase.new_attribute`.<sup>24</sup> The attribute

<sup>24</sup>Many thanks, again, to Hans Hagen for making this part configurable!

identifiers are prefixed “luaotfload@” to avoid name clashes.

```
99
100 do
101     local new_attribute    = luatexbase.new_attribute
102     local the_attributes   = luatexbase.attributes
103
104     attributes = attributes or { }
105
106     attributes.private = function (name)
107         local attr    = "luaotfload@" .. name --- used to be: "otfl@"
108         local number = the_attributes[attr]
109         if not number then
110             number = new_attribute(attr)
111         end
112         return number
113     end
114 end
115
```

These next lines replicate the behavior of `luatex-fonts.lua`.

```
116
117 local context_environment = { }
118
119 local push_namespaces = function ()
120     log("push namespace for font loader")
121     local normalglobal = { }
122     for k, v in next, _G do
123         normalglobal[k] = v
124     end
125     return normalglobal
126 end
127
128 local pop_namespaces = function (normalglobal, isolate)
129     if normalglobal then
130         local _G = _G
131         local mode = "non-destructive"
132         if isolate then mode = "destructive" end
133         log("pop namespace from font loader -- " .. mode)
134         for k, v in next, _G do
135             if not normalglobal[k] then
136                 context_environment[k] = v
137                 if isolate then
138                     _G[k] = nil
139                 end
140             end
141         end
142         for k, v in next, normalglobal do
143             _G[k] = v
144         end
145         -- just to be sure:
```

```

146         setmetatable(context_environment, _G)
147     else
148         log("irrecoverable error during pop_namespace: no globals to restore")
149         os.exit()
150     end
151 end
152
153 luaotfload.context_environment = context_environment
154 luaotfload.push_namespaces     = push_namespaces
155 luaotfload.pop_namespaces     = pop_namespaces
156
157 local our_environment = push_namespaces()
158

```

The font loader requires that the attribute with index zero be zero. We happily oblige. (Cf. luatex-fonts-nod.lua.)

```

159
160 tex.attribute[0] = 0
161

```

Now that things are sorted out we can finally load the fontloader.

```

162
163 loadmodule"merged.lua"
164 ---loadmodule"font-odv.lua" --- <= Devanagari support from Context
165
166 if fonts then
167
168     if not fonts._merge_loaded_message_done_ then
169         log ["I am using the merged version of 'luaotfload.lua' here.]]
170         log [[ If you run into problems or experience unexpected]]
171         log [[ behaviour, and if you have ConTeXt installed you can try]]
172         log [[ to delete the file 'luaotfload-merged.lua' as I might]]
173         log [[ then use the possibly updated libraries. The merged]]
174         log [[ version is not supported as it is a frozen instance.]]
175         log [[ Problems can be reported to the ConTeXt mailing list.]]
176     end
177     fonts._merge_loaded_message_done_ = true
178
179 else--- the loading sequence is known to change, so this might have to
180     --- be updated with future updates!
181     --- do not modify it though unless there is a change to the merged
182     --- package!
183     loadmodule("l-lua.lua")
184     loadmodule("l-lpeg.lua")
185     loadmodule("l-function.lua")
186     loadmodule("l-string.lua")
187     loadmodule("l-table.lua")
188     loadmodule("l-io.lua")
189     loadmodule("l-file.lua")
190     loadmodule("l-boolean.lua")

```

```

191     loadmodule("l-math.lua")
192     loadmodule("util-str.lua")
193     loadmodule('luatex-basics-gen.lua')
194     loadmodule('data-con.lua')
195     loadmodule('luatex-basics-nod.lua')
196     loadmodule('font-ini.lua')
197     loadmodule('font-con.lua')
198     loadmodule('luatex-fonts-enc.lua')
199     loadmodule('font-cid.lua')
200     loadmodule('font-map.lua')
201     loadmodule('luatex-fonts-syn.lua')
202     loadmodule('luatex-fonts-tfm.lua')
203     loadmodule('font-oti.lua')
204     loadmodule('font-otf.lua')
205     loadmodule('font-otb.lua')
206     loadmodule('node-inj.lua')
207     loadmodule('font-ota.lua')
208     loadmodule('font-otn.lua')
209     loadmodule('font-otp.lua')--- since 2013-04-23
210     loadmodule('luatex-fonts-lua.lua')
211     loadmodule('font-def.lua')
212     loadmodule('luatex-fonts-def.lua')
213     loadmodule('luatex-fonts-ext.lua')
214     loadmodule('luatex-fonts-cbk.lua')
215 end --- non-merge fallback scope
216

```

Here we adjust the globals created during font loader initialization. If the second argument to *pop\_namespaces()* is true this will restore the state of *\_G*, eliminating every global generated since the last call to *push\_namespaces()*. At the moment we see no reason to do this, and since the font loader is considered an essential part of *luatex* as well as a very well organized piece of code, we happily concede it the right to add to *\_G* if needed.

```

217
218 pop_namespaces(our_environment, false)-- true)
219
220 log("fontloader loaded in %0.3f seconds", os.gettimeofday()-starttime)
221

```

### 9.3 Callbacks

After the fontloader is ready we can restore the callback trap from *luatexbase*.

```

222
223 callback.register = trapped_register
224

```

We do our own callback handling with the means provided by *luatexbase*. Note: *pre\_linebreak\_filter* and *hpack\_filter* are coupled in ConT<sub>E</sub>Xt in the concept of *node processor*.

```

225
226 add_to_callback("pre_linebreak_filter",
227                 nodes.simple_font_handler,
228                 "luaotfload.node_processor",
229                 1)
230 add_to_callback("hpack_filter",
231                 nodes.simple_font_handler,
232                 "luaotfload.node_processor",
233                 1)
234 add_to_callback("find_vf_file",
235                 find_vf_file, "luaotfload.find_vf_file")
236
237 loadmodule"override.lua"    --- "luat-ovr"
238
239 logs.set_loglevel(config.luaotfload.loglevel)
240

```

Now we load the modules written for luaotfload.

```

241 loadmodule"loaders.lua"    --- "font-pfb" new in 2.0, added 2011
242 loadmodule"database.lua"   --- "font-nms"
243 loadmodule"colors.lua"     --- "font-clr"
244

```

Relying on the name: resolver for everything has been the source of permanent trouble with the database. With the introduction of the new syntax parser we now have enough granularity to distinguish between the X<sub>Y</sub>TeX emulation layer and the genuine name: and file: lookups of LuaTeX-Fonts. Another benefit is that we can now easily plug in or replace new lookup behaviors if necessary. The name resolver remains untouched, but it calls *fonts.names.resolve()* internally anyways (see luaotfload-database.lua).

```

245
246 local request_resolvers = fonts.definers.resolvers
247 local formats           = fonts.formats -- nice table; does lowercasing ...
248 formats.ofm             = "type1"
249

```

luaotfload promises easy access to system fonts. Without additional precautions, this cannot be achieved by kpathsea alone, because it searches only the texmf directories by default. Although it is possible for kpathsea to include extra paths by adding them to the OSFONTDIR environment variable, this is still short of the goal »*it just works!*«. When building the font database luaotfload scans system font directories anyways, so we already have all the information for looking sytem fonts. With the release version 2.2 the file names are indexed in the database as well and we are ready to resolve file: lookups this way. Thus we no longer need to call the kpathsea library in most cases when looking up font files, only when generating the database.

```

250
251 local resolvefile = fonts.names.crude_file_lookup
252 --local resolvefile = fonts.names.crude_file_lookup_verbose
253
254 request_resolvers.file = function (specification)

```

```

255     local name      = resolvefile(specification.name)
256     local suffix    = file.suffix(name)
257     if formats[suffix] then
258         specification.forced = suffix
259         specification.name   = file.removesuffix(name)
260     else
261         specification.name = name
262     end
263 end
264

```

We classify as anon: those requests that have neither a prefix nor brackets. According to Khaled<sup>25</sup> they are the X<sub>Y</sub>TeX equivalent of a name: request, so we will be treating them as such.

```

265
266 --request_resolvers.anon = request_resolvers.name
267

```

There is one drawback, though. This syntax is also used for requesting fonts in Type1 (*TFM*, *OFM*) format. These are essentially file: lookups and must be caught before the name: resolver kicks in, lest they cause the database to update. Even if we were to require the file: prefix for all Type1 requests, tests have shown that certain fonts still include further fonts (e. g. omlgcb.ofm will ask for omsecob.tfm) *using the old syntax*. For this reason, we introduce an extra check with an early return.

```

268
269 local type1_formats = { "tfm", "ofm", }
270
271 request_resolvers.anon = function (specification)
272     local name = specification.name
273     for i=1, #type1_formats do
274         local format = type1_formats[i]
275         if resolvers.findfile(name, format) then
276             specification.name = file.addsuffix(name, format)
277             return
278         end
279     end
280     --- under some weird circumstances absolute paths get
281     --- passed to the definer; we have to catch them
282     --- before the name: resolver misinterprets them.
283     name = specification.specification
284     local exists, _ = lfsisfile(name)
285     if exists then --- garbage; we do this because we are nice,
286                   --- not because it is correct
287         logs.names_report("log", 1, "load", "file %q exists", name)
288         logs.names_report("log", 1, "load",
289             "... overriding borked anon: lookup with path: lookup")
290         specification.name = name
291         request_resolvers.path(specification)

```

<sup>25</sup><https://github.com/phi-gamma/luatofload/issues/4#issuecomment-17090553>.

```

292         return
293     end
294     request_resolvers.name(specification)
295 end
296

```

Prior to version 2.2, luaotfload did not distinguish file: and path: lookups, causing complications with the resolver. Now we test if the requested name is an absolute path in the file system, otherwise we fall back to the file: lookup.

```

297
298 request_resolvers.path = function (specification)
299     local name      = specification.name
300     local exists, _ = lfs.isfile(name)
301     if not exists then -- resort to file: lookup
302         logs.names_report("log", 1, "load",
303             "path lookup of %q unsuccessful, falling back to file:",
304             name)
305         request_resolvers.file(specification)
306     else
307         local suffix = file.suffix(name)
308         if formats[suffix] then
309             specification.forced = suffix
310             specification.name   = file.removesuffix(name)
311         else
312             specification.name = name
313         end
314     end
315 end
316

```

**EXPERIMENTAL:** kpse-only resolver, for those who can do without system fonts.

```

317
318 request_resolvers.kpse = function (specification)
319     local name      = specification.name
320     local suffix     = file.suffix(name)
321     if suffix and formats[suffix] then
322         name = file.removesuffix(name)
323         if resolvers.findfile(name, suffix) then
324             specification.forced = suffix
325             specification.name   = name
326             return
327         end
328     end
329     for t, format in next, formats do --- brute force
330         if kpse.find_file (name, format) then
331             specification.forced = t
332             specification.name   = name
333             return
334         end
335     end

```



```

336 end
337

```

Also **EXPERIMENTAL**: custom file resolvers via callback.

```

338 create_callback("luaotfload.resolve_font", "simple", dummy_function)
339
340 request_resolvers.my = function (specification)
341     call_callback("luaotfload.resolve_font", specification)
342 end
343

```

We create a callback for patching fonts on the fly, to be used by other packages. It initially contains the empty function that we are going to override below.

```

344
345 create_callback("luaotfload.patch_font", "simple", dummy_function)
346

```

#### 9.4 *ConTeXt* override

We provide a simplified version of the original font definition callback.

```

347
348 local read_font_file = fonts.definers.read
349
350 --- spec -> size -> id -> tfmdata
351 local patch_defined_font = function (specification, size, id)
352     local tfmdata = read_font_file(specification, size, id)
353     if type(tfmdata) == "table" and tfmdata.shared then
354         --- We need to test for the "shared" field here
355         --- or else the fontspec capheight callback will
356         --- operate on tfm fonts.
357         call_callback("luaotfload.patch_font", tfmdata, specification)
358     end
359     return tfmdata
360 end
361
362 reset_callback "define_font"
363

```

Finally we register the callbacks.

```

364
365 local font_definer = config.luaotfload.definer
366
367 if font_definer == "generic" then
368     add_to_callback("define_font",
369         fonts.definers.read,
370         "luaotfload.define_font",
371         1)
372 elseif font_definer == "patch" then
373     add_to_callback("define_font",

```

```

374             patch_defined_font,
375             "luaotfload.define_font",
376             1)
377 end
378
379 loadmodule"features.lua"    --- contains what was "font-ltx" and "font-otc"
380 loadmodule"extralibs.lua"  --- load additional Context libraries
381 loadmodule"auxiliary.lua"  --- additionally high-level functionality (new)
382
383 luaotfload.aux.start_rewrite_fontname () --- to be migrated to fontspec
384
385 -- vim:tw=71:sw=4:ts=4:expandtab
386

```

## 10 luaotfload.sty

Classical Plain+~~La~~<sup>TeX</sup> package initialization.

```

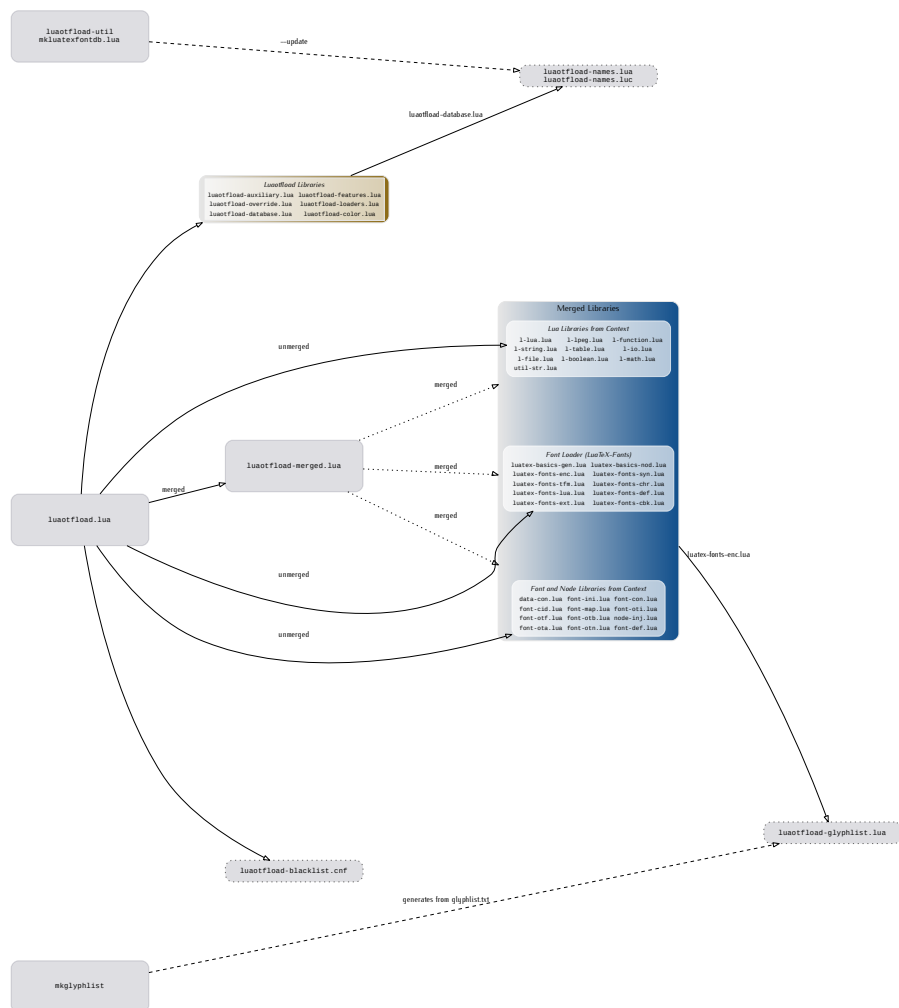
387 \csname ifluaotfloadloaded\endcsname
388 \let\ifluaotfloadloaded\endinput
389 \bgroup\expandafter\expandafter\expandafter\egroup
390 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
391   \input luatexbase.sty
392 \else
393   \NeedsTeXFormat{LaTeX2e}
394   \ProvidesPackage{luaotfload}%
395     [2013/07/23 v2.3b OpenType layout system]
396   \RequirePackage{luatexbase}
397 \fi
398 \ifnum\luatexversion<76
399   %% here some deprecation warning would be in order
400   \RequireLuaModule{lualibs}
401   \RequireLuaModule{luaotfload-legacy}
402 \else
403   \RequireLuaModule{luaotfload}
404 \fi
405 \endinput

```

$\langle \text{definition} \rangle$	::= $\backslash \text{font}$ , $\text{CSNAME}$ , '=', $\langle \text{font request} \rangle$ , [ $\langle \text{size} \rangle$ ] ;
$\langle \text{size} \rangle$	::= 'at', $\text{DIMENSION}$ ;
$\langle \text{font request} \rangle$	::= 'n', $\langle \text{unquoted font request} \rangle$ 'n'   't', $\langle \text{unquoted font request} \rangle$ '}'   $\langle \text{unquoted font request} \rangle$ ;
$\langle \text{unquoted font request} \rangle$	::= $\langle \text{specification} \rangle$ , [ ':' , $\langle \text{feature list} \rangle$ ]   '[', $\langle \text{path lookup} \rangle$ ']', [ [ ':' ], $\langle \text{feature list} \rangle$ ] ;
$\langle \text{specification} \rangle$	::= $\langle \text{prefixed spec} \rangle$ , [ $\langle \text{subfont no} \rangle$ ], { $\langle \text{modifier} \rangle$ }   $\langle \text{anon lookup} \rangle$ , { $\langle \text{modifier} \rangle$ } ;
$\langle \text{prefixed spec} \rangle$	::= 'file:', $\langle \text{file lookup} \rangle$   'name:', $\langle \text{name lookup} \rangle$ ;
$\langle \text{file lookup} \rangle$	::= { $\langle \text{name character} \rangle$ } ;
$\langle \text{name lookup} \rangle$	::= { $\langle \text{name character} \rangle$ } ;
$\langle \text{anon lookup} \rangle$	::= $\text{TFMNAME}$   $\langle \text{name lookup} \rangle$ ;
$\langle \text{path lookup} \rangle$	::= { $\text{ALL\_CHARACTERS}$ - ']' } ;
$\langle \text{modifier} \rangle$	::= '/', ('I'   'B'   'BI'   'IB'   'S=', { $\text{DIGIT}$ } ) ;
$\langle \text{subfont no} \rangle$	::= '(', { $\text{DIGIT}$ }, ')' ;
$\langle \text{feature list} \rangle$	::= $\langle \text{feature expr} \rangle$ , { ' ; ', $\langle \text{feature expr} \rangle$ } ;
$\langle \text{feature expr} \rangle$	::= $\text{FEATURE\_ID}$ , '=', $\text{FEATURE\_VALUE}$   $\langle \text{feature switch} \rangle$ , $\text{FEATURE\_ID}$ ;
$\langle \text{feature switch} \rangle$	::= '+'   '-' ;
$\langle \text{name character} \rangle$	::= $\text{ALL\_CHARACTERS}$ - ( '('   '/'   ':' ) ;

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:  $\text{FEATURE\_ID}$  for a valid font feature name and  $\text{FEATURE\_VALUE}$  for the corresponding value.  $\text{TFMNAME}$  is the name of a TFM file.  $\text{DIGIT}$  again refers to bytes 48–57, and  $\text{ALL\_CHARACTERS}$  to all byte values.  $\text{CSNAME}$  and  $\text{DIMENSION}$  are the T<sub>E</sub>X concepts.

Figure 2: Schematic of the files in Luaotfload



The GPL requires the complete license text to be distributed along with the code. I recommend the canonical source, instead: <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>. But if you insist on an included copy, here it is. You might want to zoom in.

## GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things. To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights, and we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole or as close to all third parties under the terms of this License.
- If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole

which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or
- Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or
- Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection 1 above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property rights claims or to contest validity of any such claims, this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## NO WARRANTY

12. *BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.*

13. *IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR RE-DISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITHIN ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

## END OF TERMS AND CONDITIONS

## APPENDIX: HOW TO APPLY THESE TERMS TO YOUR NEW PROGRAMS

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does.  
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.  
If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

GNUversion 69, Copyright (C) yyyy name of author  
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  
This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yooyah, Inc., hereby disclaims all copyright interest in the program  
"Gnomovision" (which makes passes at compilers) written by James  
Hacker.

signature of Ty Coon, 1 April 1989  
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.