The luaotfload package

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

2013/05/05 v2.2

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.

Contents

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[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 XƎTEX engine.

Unlike XƎTEX, 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.

2 *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.

2.1 *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:`. It determines whether the font loader should interpret the request as a *file name* or *font name*, respectively, which again influences

---

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

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. 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 4 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`.

2.2 *XeTeX 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.

2.3 *Examples*

2.3.1 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*[2] in its condensed variant can be loaded as follows:

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

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

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

2.3.2 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}

...
```

2.3.3 Modifiers

If the entire *Iwona* family[3] 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:

---

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

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

```
\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
```

3 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 (XƎTEX style), they must precede ⟨*font features*⟩.

The element ⟨*font features*⟩ is a semicolon-separated list of feature tags[4] 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 TEX ligature and kerning mechanisms. Supporting only non-contextual substitutions and kerning

---

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

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

language

An OpenType language system identifier,[6] defaulting to `dflt`.

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.[7]

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.[8] For both, only the set default is predefined.

---

[5] 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.

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

[7] Cf. `http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html`.

[8] 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)}
```

For example, to enable default protrusion[9]:

```
\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:

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:

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

[10]

itlc Computes italic correction values (active by default).

4 *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.[11]

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.

```
inspect(fonts.expansions.setups.default)}
```

at some point after loading `luaotfload.sty`.

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

[10] 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.

[11] 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.

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`

4.1 *luaotfload-tool / mkluatexfontdb.lua*[12]

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
```

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

4.2 *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.

---

[12] The script may be named just `mkluatexfontdb` in your distribution.

[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`.

4.3 *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]

`luaotfload-tool --help` will list the available command line switches, including some not discussed in detail here.

4.4 *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**,[15] 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.

---

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

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

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.
```

5 *Files from ConTEXt and LuaTEX-Fonts*

luaotfload relies on code originally written by Hans Hagen[16] for and tested with ConTEXt. It integrates the font loader as distributed in the **LuaTEX-Fonts** package. The original 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[17] 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 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 24 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 too developed by Hans Hagen.[18] 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.
    - `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`
- The *font loader* itself. These files have been written for LuaTEX-Fonts and they are distributed along with **luaotfload**.

---

[16] The creator of the ConTEXt format.

[17] It covers, however, to some extent the functionality of the **lualibs** package.

[18] `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.

  - `luatex-basics-gen.lua`
  - `luatex-basics-nod.lua`
  - `luatex-fonts-enc.lua`
  - `luatex-fonts-syn.lua`
  - `luatex-fonts-tfm.lua`
  - `luatex-fonts-chr.lua`
  - `luatex-fonts-lua.lua`
  - `luatex-fonts-def.lua`
  - `luatex-fonts-ext.lua`
  - `luatex-fonts-cbk.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-oti.lua`
  - `font-otf.lua`
  - `font-otb.lua`
  - `node-inj.lua`
  - `font-ota.lua`
  - `font-otn.lua`
  - `font-def.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 ConTEXt (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 ConTEXt 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 `font-age.lua`, which is automatically generated by the script `mkglyphlist`.[19] There is a make target **glyphs** that will create a fresh `font-age.lua` 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-lib-dir.lua` – `l-dir` from ConTEXt; contains functionality required by `luaotfload-font-nms.lua`.
- `luaotfload-override.lua` – overrides the ConTEXt 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.

---

[19] See `luaotfload-font-enc.lua`. The hard-coded file name is why the file lacks the `luaotfload-` prefix.

- `luaotfload-auxiliary.lua` – access to internal functionality for package authors (proposals for additions welcome).

6 *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 `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;
```

II Implementation

7 `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 initilization. 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 LuaTEX-Fonts self-contained and encapsulate it, and especially due to his willingness to incorporate our suggestions.

```
luaotfload                  = luaotfload or {}
local luaotfload            = luaotfload

config                              = config or { }
```

```
config.luaotfload                 = config.luaotfload or { }
------.luaotfload.resolver        = config.luaotfload.resolver  or ”normal”
config.luaotfload.resolver        = config.luaotfload.resolver  or ”cached”
config.luaotfload.definer         = config.luaotfload.definer   or ”patch”
config.luaotfload.loglevel        = config.luaotfload.loglevel  or 1
config.luaotfload.color_callback  = config.luaotfload.color_callback  or ”pre_linebreak_filter”
--luaotfload.prefer_merge     = config.luaotfload.prefer_merge or true

luaotfload.module = {
    name          = ”luaotfload”,
    version       = 2.2,
    date          = ”2013/04/29”,
    description   = ”OpenType layout system.”,
    author        = ”Elie Roux & Hans Hagen”,
    copyright     = ”Elie Roux”,
    license       = ”GPL v2.0”
}

local luatexbase = luatexbase

local type, next       = type, next
local setmetatable     = setmetatable
local find_file        = kpse.find_file
local lfsisfile        = lfs.isfile
local stringfind       = string.find
local stringformat     = string.format
local stringmatch      = string.match
local stringsub        = string.sub

local add_to_callback, create_callback =
      luatexbase.add_to_callback, luatexbase.create_callback
local reset_callback, call_callback =
      luatexbase.reset_callback, luatexbase.call_callback

local dummy_function = function () end

```

No final decision has been made on how to handle font definition. At the moment, there are three candidates: The generic callback as hard-coded in the font loader, the old wrapper, and a simplified version of the latter (patch) that does nothing besides applying font patches.

```

luaotfload.font_definer = ”patch” --- | “generic” | “old”

local error, warning, info, log =
    luatexbase.provides_module(luaotfload.module)

luaotfload.error        = error
luaotfload.warning      = warning
luaotfload.info         = info
luaotfload.log          = log

```

We set the minimum version requirement for LuaTEX 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.[20]

```

local luatex_version = 76

if tex.luatexversion < luatex_version then
    warning(”LuaTeX v%.2f is old, v%.2f is recommended.”,
             tex.luatexversion/100,
             luatex_version   /100)
end

```

7.1 *Module loading*

We load the files imported from ConTEXt with this function. It automatically prepends the prefix `luaotfload-` to its argument, so we can refer to the files with their actual ConTEXt name.

```

local fl_prefix = ”luaotfload” -- “luatex” for luatex-plain
local loadmodule = function (name)
    require(fl_prefix ..”-”..name)
end

```

Before TEXLive 2013 version, LuaTEX 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 questionable as they are not necessary since TEXLive 2013. They should be removed in the next version.

```
local Cs, P, lpegmatch = lpeg.Cs, lpeg.P, lpeg.match

local p_dot, p_slash = P”.”,  P”/”
local p_suffix       = (p_dot * (1 - p_dot - p_slash)^1 * P(-1)) / ””
local p_removesuffix = Cs((p_suffix + 1)^1)

local find_vf_file = function (name)
    local fullname = find_file(name, ”ovf”)
    if not fullname then
        --fullname = find_file(file.removesuffix(name), ”ovf”)
        fullname = find_file(lpegmatch(p_removesuffix, name), ”ovf”)
```

---

[20] 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`.

```
    end
    if fullname then
        log(”loading virtual font file %s.”, fullname)
    end
    return fullname
end

```

7.2 *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 LuaTEX-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 ConTEXt 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 **lualibs** 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.

```

local starttime = os.gettimeofday()

local trapped_register  = callback.register
callback.register       = dummy_function

```

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***.[21] The attribute identifiers are prefixed “`luaotfload@`” to avoid name clashes.

```

do
    local new_attribute    = luatexbase.new_attribute
    local the_attributes   = luatexbase.attributes

    attributes = attributes or { }

    attributes.private = function (name)
        local attr   = ”luaotfload@” .. name --- used to be: “otfl@”
        local number = the_attributes[attr]
        if not number then
```

---

[21] Many thanks, again, to Hans Hagen for making this part configurable!

```
            number = new_attribute(attr)
        end
        return number
    end
end

```

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

```

local context_environment = { }

local push_namespaces = function ()
    log(”push namespace for font loader”)
    local normalglobal = { }
    for k, v in next, _G do
        normalglobal[k] = v
    end
    return normalglobal
end

local pop_namespaces = function (normalglobal, isolate)
    if normalglobal then
        local _G = _G
        local mode = ”non-destructive”
        if isolate then mode = ”destructive” end
        log(”pop namespace from font loader -- ” .. mode)
        for k, v in next, _G do
            if not normalglobal[k] then
                context_environment[k] = v
                if isolate then
                    _G[k] = nil
                end
            end
        end
        for k, v in next, normalglobal do
            _G[k] = v
        end
        -- just to be sure:
        setmetatable(context_environment,_G)
    else
        log(”irrecoverable error during pop_namespace: no globals to restore”)
        os.exit()
    end
end

luaotfload.context_environment  = context_environment
luaotfload.push_namespaces      = push_namespaces
luaotfload.pop_namespaces       = pop_namespaces

local our_environment = push_namespaces()
```

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

```

tex.attribute[0] = 0

```

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

```

loadmodule”merged.lua”
---loadmodule”font-odv.lua” --- <= Devanagari support from Context

if fonts then

    if not fonts._merge_loaded_message_done_ then
        --- a program talking first person -- HH sure believes in strong AI ...
        log[[“I am using the merged version of ’luaotfload.lua’ here. If]]
        log[[ you run into problems or experience unexpected behaviour,]]
        log[[ and if you have ConTeXt installed you can try to delete the]]
        log[[ file ’luaotfload-font-merged.lua’ as I might then use the]]
        log[[ possibly updated libraries. The merged version is not]]
        log[[ supported as it is a frozen instance. Problems can be]]
        log[[ reported to the ConTeXt mailing list.”]]
    end
    fonts._merge_loaded_message_done_ = true

else--- the loading sequence is known to change, so this might have to
    --- be updated with future updates!
    --- do not modify it though unless there is a change to the merged
    --- package!
    loadmodule(”l-lua.lua”)
    loadmodule(”l-lpeg.lua”)
    loadmodule(”l-function.lua”)
    loadmodule(”l-string.lua”)
    loadmodule(”l-table.lua”)
    loadmodule(”l-io.lua”)
    loadmodule(”l-file.lua”)
    loadmodule(”l-boolean.lua”)
    loadmodule(”l-math.lua”)
    loadmodule(”util-str.lua”)
    loadmodule(’luatex-basics-gen.lua’)
    loadmodule(’data-con.lua’)
    loadmodule(’luatex-basics-nod.lua’)
    loadmodule(’font-ini.lua’)
    loadmodule(’font-con.lua’)
    loadmodule(’luatex-fonts-enc.lua’)
    loadmodule(’font-cid.lua’)
    loadmodule(’font-map.lua’)
    loadmodule(’luatex-fonts-syn.lua’)
    loadmodule(’luatex-fonts-tfm.lua’)
    loadmodule(’font-oti.lua’)
    loadmodule(’font-otf.lua’)
    loadmodule(’font-otb.lua’)
    loadmodule(’node-inj.lua’)
    loadmodule(’font-ota.lua’)
    loadmodule(’font-otn.lua’)
    loadmodule(’font-otp.lua’)--- since 2013-04-23
    loadmodule(’luatex-fonts-lua.lua’)
    loadmodule(’font-def.lua’)
    loadmodule(’luatex-fonts-def.lua’)
    loadmodule(’luatex-fonts-ext.lua’)
    loadmodule(’luatex-fonts-cbk.lua’)
end --- non-merge fallback scope

```

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.

```

pop_namespaces(our_environment, false)-- true)

log(”fontloader loaded in %0.3f seconds”, os.gettimeofday()-starttime)

```

7.3 *Callbacks*

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

```

callback.register = trapped_register

```

We do our own callback handling with the means provided by luatexbase. Note: ***pre_linebreak_filter*** and ***hpack_filter*** are coupled in ConTeXt in the concept of *node processor*.

```

add_to_callback(”pre_linebreak_filter”,
                nodes.simple_font_handler,
                ”luaotfload.node_processor”,
                1)
add_to_callback(”hpack_filter”,
                nodes.simple_font_handler,
                ”luaotfload.node_processor”,
                1)
add_to_callback(”find_vf_file”,
                find_vf_file, ”luaotfload.find_vf_file”)
```

```
loadmodule”lib-dir.lua”   --- required by luaofload-database.lua
loadmodule”override.lua”  --- “luat-ovr”

logs.set_loglevel(config.luaotfload.loglevel)

```

Now we load the modules written for **luaotfload**.

```
loadmodule”loaders.lua”    --- “font-pfb” new in 2.0, added 2011
loadmodule”database.lua”   --- “font-nms”
loadmodule”colors.lua”     --- “font-clr”

```

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 XeTeX 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`).

```

local request_resolvers   = fonts.definers.resolvers
local formats             = fonts.formats
formats.ofm               = ”type1”

```

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.

```
request_resolvers.file = function (specification)
    local found = fonts.names.crude_file_lookup(specification.name)
    --local found = fonts.names.crude_file_lookup_verbose(specification.name)
    specification.name = found[1]
    --if format then specification.forced = format end
end

```

We classify as `anon:` those requests that have neither a prefix nor brackets. According to Khaled[22] they are the XeTeX equivalent of a `name:` request, so we will be treating them as such.

```

--request_resolvers.anon = request_resolvers.name

```

---

[22] https://github.com/phi-gamma/luaotfload/issues/4#issuecomment-17090553.

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.

```
local type1_formats = { ”tfm”, ”ofm”, }

request_resolvers.anon = function (specification)
    local name = specification.name
    for i=1, #type1_formats do
        local format = type1_formats[i]
        if resolvers.findfile(name, format) then
            specification.name = file.addsuffix(name, format)
            return
        end
    end
    request_resolvers.name(specification)
end
```

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.

```
request_resolvers.path = function (specification)
    local exists, _ = lfsisfile(specification.name)
    if not exists then -- resort to file: lookup
        request_resolvers.file(specification)
    end
end
```

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.

```
create_callback(”luaotfload.patch_font”, ”simple”, dummy_function)
```

7.4 *ConTEXt override*

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

```
local read_font_file = fonts.definers.read

--- spec -> size -> id -> tmfdata
local patch_defined_font = function (specification, size, id)
    local tfmdata = read_font_file(specification, size, id)
    if type(tfmdata) == ”table” and tfmdata.shared then
```

```
        --- We need to test for the “shared” field here
        --- or else the fontspec capheight callback will
        --- operate on tfm fonts.
        call_callback(”luaotfload.patch_font”, tfmdata)
    end
    return tfmdata
end

caches.compilemethod = ”both”

reset_callback(”define_font”)

```

Finally we register the callbacks.

```

local font_definer = config.luaotfload.definer

if font_definer == ”generic”  then
  add_to_callback(”define_font”,
                  fonts.definers.read,
                  ”luaotfload.define_font”,
                  1)
elseif font_definer == ”patch”  then
  add_to_callback(”define_font”,
                  patch_defined_font,
                  ”luaotfload.define_font”,
                  1)
end

loadmodule”features.lua”    --- contains what was “font-ltx” and “font-otc”
loadmodule”auxiliary.lua”   --- additionaly high-level functionality (new)

-- vim:tw=71:sw=4:ts=4:expandtab
```

8 `luaotfload.sty`

Classical Plain+LaTeX package initialization.

```
\csname ifluaotfloadloaded\endcsname
\let\ifluaotfloadloaded\endinput
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname ProvidesPackage\endcsname\relax
  \input luatexbase.sty
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luaotfload}%
    [2013/04/16 v2.2 OpenType layout system]
  \RequirePackage{luatexbase}
```

```
\fi
\RequireLuaModule{luaotfload}
\endinput
```

⟨*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*⟩ ::= ‘`file:`’, ⟨*file lookup*⟩
| ‘`name:`’, ⟨*name lookup*⟩ ;

⟨*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*⟩ } ;

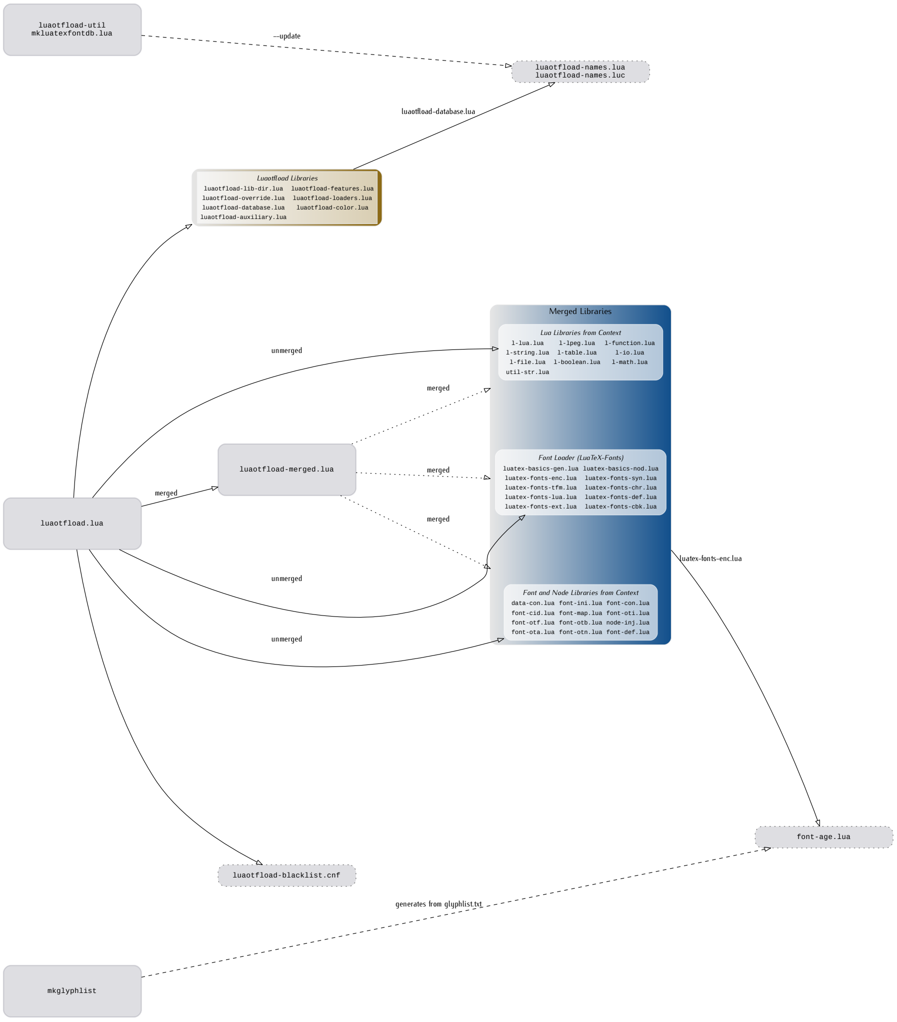
⟨*feature expr*⟩ ::= *FEATURE_ID*, ‘`=`’, *FEATURE_VALUE*
| ⟨*feature switch*⟩, *FEATURE_ID* ;

⟨*feature switch*⟩ ::= ‘`+`’ | ‘`-`’ ;

⟨*name character*⟩ ::= *ALL_CHARACTERS* - (‘`(`’ | ‘`/`’ | ‘`:`’) ;

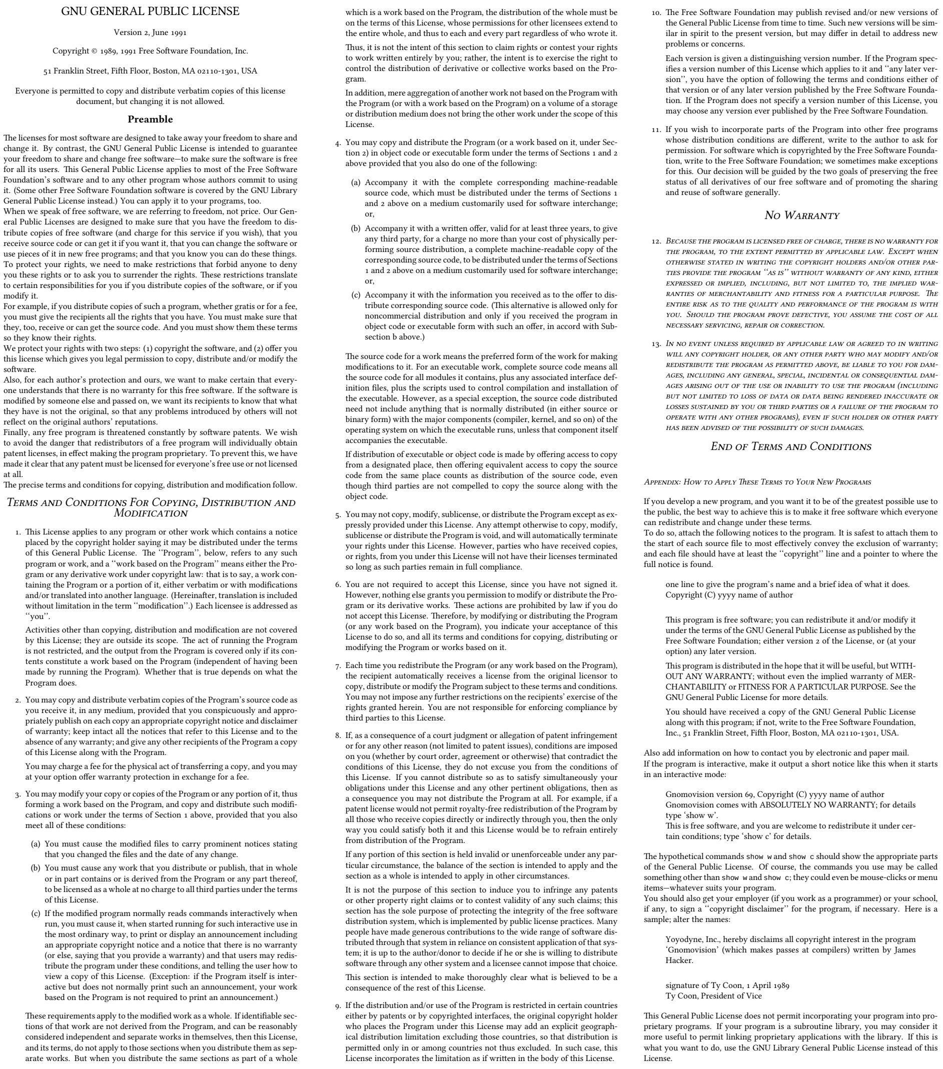
Figure 1: Font request syntax. Braces or double quotes around the *specification* rule will preserve whitespace in file names. In addition to the font style modifiers (*slash-notation*) given above, there are others that are recognized but will be silently ignored: `aat`, `icu`, and `gr`. The special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

Figure 2: Schematic of the files in Luaotfload



9 *The GNU GPL License v2*

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

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 with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, 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:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) 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 at no charge to all third parties under the terms of this License.

   (c) 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:

   (a) 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,

   (b) 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,

   (c) 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 b 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 right 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 REDISTRIBUTE 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 WITH 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:

> Gnomovision version 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:

> Yoyodyne, 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.