The luaotfload package

Elie Roux
elie.roux@telecom-bretagne.eu

2009/09/10 v1.04

Abstract

ConTEXt font loading system, providing the possibility to load OTF fonts with a lot of features, and the XeTEX font loading syntax.

Contents

1 Documentation

1.1 Introduction

Font management and installation has always been painful with TEX (and even more with LATEX). A lot of files are needed for one font (tfm, pfb, map, fd, vf), and they are limited to 256 characters. But the font world has evolved since, and new standard types of fonts have appeared, like truetype or opentype fonts. These fonts can contain a lot of characters, and have some functionalities (ligatures, old-style numbers, small capitals, etc.). They are everywhere, as the system fonts and most modern text softwares fonts are of this type. Until now the (almost) only way to use them with TEX was to use them with XeTEX.

Unlike XeTEX, LuaTEX does not provide facilities for these fonts by default, but it provides a way to hook lua code in some points of the TEX algorithm, for instance we can improve the font loading system; this is what we do in this package.

This package is quite low-level, and should be loaded directly in the macro package, like it is in ConTEXt. Sadly, Plain and LATEX are frozen and it's even impossible to adapt them to the new engines.

1.2 ConTEXt files needed

This package is a wrapper for several files taken from the ConTEXt macro package. The philosophy is to let ConTEXt do all the implementation and update these files from time to time. To do so we did not modify the files taken from ConTEXt, we only changed their names to prevent name clashes. You can thus update the font system of this package simply by updating the files taken from ConTEXt, without (theorically) changing the `.sty` file nor the main `.lua` file.

The ConTEXt files are renamed by adding the prefix `otfl-` to them (`otfl` as `OTF Load`).The files are:

- `luat-dum.lua`
- `data-con.lua`
- `node-ini.lua`
- `node-inj.lua`
- `node-fnt.lua`
- `node-dum.lua`
- `font-ini.lua`
- `font-tfm.lua`
- `font-cid.lua`
- `font-ott.lua`
- `font-otf.lua`
- `font-otd.lua`
- `font-oti.lua`
- `font-otb.lua`
- `font-otn.lua`
- `font-ota.lua`
- `font-otc.lua`
- `font-def.lua`
- `font-xtx.lua`
- `font-map.lua`
- `font-dum.lua`

1.3 Troubleshooting

If you encounter problems with some fonts, please first update to the latest version of this package before reporting a bug, as this package is under active development.

A very common problem is the lack of features for some otf fonts even when specified. It can be related to the fact that some fonts do not provide features for the `dflt` script, which is the default one in this package, so you may have to specify the script in the command line, for example:

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

2 luaotfload.lua

First some usual initializations.

```
luaotfload          = { }

luaotfload.module = {
    name          = "luaotfload",
    version       = 1.04,
    date          = "2009/09/10",
    description   = "ConTeXt font loading system.",
    author        = "Elie Roux & Hans Hagen",
    copyright     = "Elie Roux",
    license       = "CC0"
}

luatextra.provides_module(luaotfload.module)

```

We load the ConTEXt files with this function. It automatically adds the `otfl-` prefix to it, so that we call it with the actual ConTEXt name.

```


function luaotfload.loadmodule(name)
    local foundname = kpse.find_file('otfl-'..name,"tex")
    if not foundname then
      luatextra.module_error('luaotfload', string.format('file otfl-%s not found.', name))
      return
    end
    dofile(foundname)
end

```

The following functions are made to map ConTEXt functions to luaextra functions.

```

string.strip = string.stripspaces

file = fpath
```

```
file.extname = fpath.suffix

```

These are small functions that are not already in luatextra.

```

local splitters_s, splitters_m = { }, { }

function lpeg.splitat(separator,single)
    local splitter = (single and splitters_s[separator]) or splitters_m[separator]
    if not splitter then
        separator = lpeg.P(separator)
        if single then
            local other, any = lpeg.C((1 - separator)^0), lpeg.P(1)
            splitter = other * (separator * lpeg.C(any^0) + "")
            splitters_s[separator] = splitter
        else
            local other = lpeg.C((1 - separator)^0)
            splitter = other * (separator * other)^0
            splitters_m[separator] = splitter
        end
    end
    return splitter
end

function table.compact(t)
    if t then
        for k,v in next, t do
            if not next(v) then
                t[k] = nil
            end
        end
    end
end

function table.sortedhashkeys(tab) -- fast one
    local srt = { }
    for key,_ in next, tab do
        srt[#srt+1] = key
    end
    table.sort(srt)
    return srt
end

function table.reverse_hash(h)
    local r = { }
    for k,v in next, h do
        r[v] = string.lower(string.gsub(k," ",""))
    end
    return r
end
```

```
78
79 function table.reverse(t)
80     local tt = { }
81     if #t > 0 then
82         for i=#t,1,-1 do
83             tt[#tt+1] = t[i]
84         end
85     end
86     return tt
87 end
88
```

We start loading some lua files. These two are some code not used by ConTEXt at all that allow other modules to be used, it provides some low-level ConTEXt functions.

```
89
90 luaotfload.loadmodule('luat-dum.lua') -- not used in context at all
91 luaotfload.loadmodule('data-con.lua') -- maybe some day we don't need this one
92
```

This one is for node support.

```
93
94 luaotfload.loadmodule('node-ini.lua')
95
```

By default ConTEXt takes some private attributes for internal use. With Plain and LATEX we can't do so, we use `\newluaattribute`. This functions overrides a function defined in the previous module that returns the number of a private attribute. We allocate new attributes in the `.sty` file, and this function returns their number. Like this we don't need any private attribute, and this package is compatible with the others. We use the `otfl@` prefix for attributes.

```
96
97 function attributes.private(name)
98     local number = tex.attributenumber['otfl@'..name]
99     if not number then
100        luatextra.module_error('luaotfload', string.format('asking for attribute %s, but not
101    end
102    return number
103 end
104
```

Some more modules. We don't load neither `font-enc.lua` nor `font-afm.lua` as it will never be used here.

```
105
106 luaotfload.loadmodule('node-res.lua')
107 luaotfload.loadmodule('node-inj.lua')
108 luaotfload.loadmodule('node-fnt.lua')
109 luaotfload.loadmodule('node-dum.lua')
110
```

```
luaotfload.loadmodule('font-ini.lua')
luaotfload.loadmodule('font-tfm.lua')
luaotfload.loadmodule('font-cid.lua')
luaotfload.loadmodule('font-ott.lua')
luaotfload.loadmodule('font-otf.lua')
luaotfload.loadmodule('font-otd.lua')
luaotfload.loadmodule('font-oti.lua')
luaotfload.loadmodule('font-otb.lua')
luaotfload.loadmodule('font-otn.lua')
luaotfload.loadmodule('font-ota.lua')
luaotfload.loadmodule('font-otc.lua')

```

`font-def.lua` calls the function `callback.register` to register its callbacks. We override it with a dumb function so that it does not register any callback. We will register the callbacks later.

```

do
  local temp = callback.register
  callback.register = function (...)
    return
  end
  luaotfload.loadmodule('font-def.lua')
  callback.register = temp
end

luaotfload.loadmodule('font-xtx.lua')
luaotfload.loadmodule('font-map.lua')
luaotfload.loadmodule('font-dum.lua')

```

This is a small patch that prevents errors in some LaTeX files.

```

fonts.enc.known = {}

```

We have to register a function in the `find_vf_file` callback in order to make everything work.

```

function luaotfload.find_vf_file(name)
    name = file.removesuffix(file.basename(name))
    local result = kpse.find_file(name, "vf") or ""
    if result == "" then
        result = kpse.find_file(name, "ovf") or ""
    end
    return result
end

```

Finally two functions

```
function luaotfload.register_callbacks()
    callback.add('pre_linebreak_filter', nodes.simple_font_handler, 'luaotfload.pre_linebrea
    callback.add('hpack_filter',         nodes.simple_font_handler, 'luaotfload.hpack_filter
    callback.reset('define_font')
    callback.add('define_font', fonts.define.read, 'luaotfload.define_font', 1)
    callback.add('find_vf_file', luaotfload.find_vf_file, 'luaotfload.find_vf_file')
end

function luaotfload.unregister_callbacks()
    callback.remove('pre_linebreak_filter', 'luaotfload.pre_linebreak_filter')
    callback.remove('hpack_filter', 'luaotfload.hpack_filter')
    callback.remove('define_font', 'luaotfload.define_font')
    callback.remove('find_vf_file', 'luaotfload.find_vf_file')
end
```

3 luaotfload.sty

Classical Plain+LaTeX package initialization.

```
\csname ifluaotfloadloaded\endcsname
\let\ifluaotfloadloaded\endinput

\expandafter\ifx\csname ProvidesPackage\endcsname\relax
  \input luatextra.sty
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luaotfload}%
    [2009/09/10 v1.04 ConTeXt font loading system]
  \RequirePackage{luatextra}
\fi

\expandafter\edef\csname otfl@AtEnd\endcsname{%
  \catcode64 \the\catcode64\relax
}

\catcode64 11

```

The attributes are allocated here. The `otfl@` prefix is added to prevent name collision.

```

\newluaattribute\otfl@state
\newluaattribute\otfl@markbase
\newluaattribute\otfl@markdone
\newluaattribute\otfl@markmark
\newluaattribute\otfl@cursbase
\newluaattribute\otfl@curscurs
\newluaattribute\otfl@cursdone
```

```
\newluaattribute\otfl@kernpair
\newluaattribute\otfl@color

```

Two small macros to register or unregister the callbacks. Without the callbacks this package is totally turned off.

```

\def\otfl@off{
\luadirect{luaotfload.unregister_callbacks()}
}

\def\otfl@on{
\luadirect{luaotfload.register_callbacks()}
}

```

We load the `lua` file, and we turn the package on.

```

\luaUseModule{luaotfload}

\otfl@on

\otfl@AtEnd
```