

The `luatextra` package

Elie Roux

`elie.roux@telecom-bretagne.eu`

2010/08/02 v0.971

Abstract

This document is made for people wanting to understand how the package was made. For an introduction to the use of LuaTeX with the formats Plain and L^AT_EX, please read the document `luatextra-reference.pdf` that you can find in your T_EX distribution (T_EXLive from version 2009) or on the CTAN.

Warning. This package is undergoing major changes, so the documentation may sometimes be inconsistent. Also, be aware that everything may change.

Contents

1	<code>luatextra.lua</code>	1
1.1	Initialization and internal functions	1
1.2	Multiple callbacks on the <code>open_read_file</code> callback	2
1.3	Multiple callbacks on the <code>define_font</code> callback	3
2	<code>luatextra.sty</code>	6
2.1	Initializations	6
2.2	Primitives renaming	7

1 `luatextra.lua`

1.1 Initialization and internal functions

We create the `luatextra` table that will contain all the functions and variables, and we register it as a normal lua module.

```
1 module("luatextra", package.seeall)
```

We initiate the `modules` table that will contain informations about the loaded modules. And we register the `luatextra` module. The informations contained in the table describing the module are always the same, it can be taken as a template. See `luatextra.provides_module` for more details.

```

2 luatexbase.provides_module {
3   version      = 0.971,
4   name         = "luatextra",
5   date         = "2010/08/02",
6   description   = "Additional low level functions for LuaTeX",
7   author       = "Elie Roux and Manuel Pegourie-Gonnard",
8   copyright    = "Elie Roux, 2009 and Manuel Pegourie-Gonnard, 2010",
9   license      = "CC0",
10 }
11 local format = string.format

```

1.2 Multiple callbacks on the `open_read_file` callback

`luatexbase` (see documentation for details) cannot really provide a simple and reliable way of registering multiple functions in some callbacks. To be able to do so, the solution we implemented is to register one function in these callbacks, and to create "sub-callbacks" that can accept several functions. That's what we do here for the callback `open_read_file`.

`luatextra.open_read_file` This function is the one that will be registered in the callback. It calls new callbacks, that will be created later. These callbacks are:

- `pre_read_file` in which you can register a function with the signature `pre_read_file(env)`, with `env` being a table containing the fields `filename` which is the argument of the callback `open_read_file`, and `path` which is the result of `kpse.find_file`. You can put any field you want in the `env` table, you can even override the existing fields. This function is called at the very beginning of the callback, it allows for instance to register functions in the other callbacks. It is useless to add a field `reader` or `close`, as they will be overridden.
- `file_reader` is automatically registered in the `reader` callback for every file, it has the same signature.
- `file_close` is registered in the `close` callback for every file, and has the same signature.

```

12 function luatextra.open_read_file(filename)
13   local path = kpse.find_file(filename)
14   local env = {
15     ['filename'] = filename,
16     ['path'] = path,
17   }
18   luatexbase.call_callback('pre_read_file', env)
19   path = env.path
20   if not path then
21     return
22   end
23   local f = env.file

```

```

24     if not f then
25         f = io.open(path)
26         env.file = f
27     end
28     if not f then
29         return
30     end
31     env.reader = luatextra.reader
32     env.close = luatextra.close
33     return env
34 end

```

The two next functions are the one called in the `open_read_file` callback.

```

35 function luatextra.reader(env)
36     local line = (env.file):read()
37     line = luatexbase.call_callback('file_reader', env, line)
38     return line
39 end
40 function luatextra.close(env)
41     (env.file):close()
42     luatexbase.call_callback('file_close', env)
43 end

```

In the callback creation process we need to have default behaviours. Here they are. These are called only when no function is registered in the created callback. See the documentation of `luatexbase` for more details.

```

44 function luatextra.default_reader(env, line)
45     return line
46 end
47 function luatextra.default_close(env)
48     return
49 end
50 function luatextra.default_pre_read(env)
51     return env
52 end

```

1.3 Multiple callbacks on the `define_font` callback

The same principle is applied to the `define_font` callback. The main difference is that this mechanism is not applied by default. The reason is that the callback most people will register in the `define_font` callback is the one from ConTeXt allowing the use of OT fonts. When the code will be more adapted (not so soon certainly), this mechanism will certainly be used, as it allows more flexibility in the font syntax, the OT font load mechanism, etc.

The callbacks we register here are the following ones:

- `font_syntax` that takes a table with the fields `asked_name`, `name` and `size`, and modifies this table to add more information. It must add at least a `path`

field. The structure of the final table is not precisely defined, as it can vary from one syntax to another.

- `open_otf_font` takes the previous table, and must return a valid font structure as described in the LuaTeX manual.
- `post_font_opening` takes the final font table and can modify it, before this table is returned to the `define_font` callback.

But first, we acknowledge the fact that `fontforge` has been renamed to `fontloader`. This check allows older versions of LuaTeX to use `fontloader`.

As this mechanism is not loaded by default and certainly won't be until version 1.0 of LuaTeX, we don't document it further. See the documentation of `luatextra.sty` (macro `\ltextra@RegisterFontCallback`) to know how to load this mechanism anyway.

```

53 do
54   if tex.luatexversion < 36 then
55     fontloader = fontforge
56   end
57 end
58 function luatextra.find_font(name)
59   local types = {'ofm', 'ovf', 'opentype fonts', 'truetype fonts'}
60   local path = kpse.find_file(name)
61   if path then return path end
62   for _,t in pairs(types) do
63     path = kpse.find_file(name, t)
64     if path then return path end
65   end
66   return nil
67 end
68 function luatextra.font_load_error(error)
69   luatextra.module_warning('luatextra', string.format('%s\nloading lmr10 instead...', error))
70 end
71 function luatextra.load_default_font(size)
72   return font.read_tfm("lmr10", size)
73 end
74 function luatextra.define_font(name, size)
75   if (size < 0) then size = (- 655.36) * size end
76   local fontinfos = {
77     asked_name = name,
78     name = name,
79     size = size
80   }
81   callback.call('font_syntax', fontinfos)
82   name = fontinfos.name
83   local path = fontinfos.path
84   if not path then
85     path = luatextra.find_font(name)
86     fontinfos.path = luatextra.find_font(name)

```

```

87     end
88     if not path then
89         luatextra.font_load_error("unable to find font "..name)
90         return luatextra.load_default_font(size)
91     end
92     if not fontinfos.filename then
93         fontinfos.filename = file.basename(path)
94     end
95     local ext = file.suffix(path)
96     local f
97     if ext == 'tfm' or ext == 'ofm' then
98         f = font.read_tfm(name, size)
99     elseif ext == 'vf' or ext == 'ovf' then
100         f = font.read_vf(name, size)
101     elseif ext == 'ttf' or ext == 'otf' or ext == 'ttc' then
102         f = luatexbase.call_callback('open_otf_font', fontinfos)
103     else
104         luatextra.font_load_error("unable to determine the type of font "..name)
105         f = luatextra.load_default_font(size)
106     end
107     if not f then
108         luatextra.font_load_error("unable to load font "..name)
109         f = luatextra.load_default_font(size)
110     end
111     luatexbase.call_callback('post_font_opening', f, fontinfos)
112     return f
113 end
114 function luatextra.default_font_syntax(fontinfos)
115     return
116 end
117 function luatextra.default_open_otf(fontinfos)
118     return nil
119 end
120 function luatextra.default_post_font(f, fontinfos)
121     return true
122 end
123 function luatextra.register_font_callback()
124     luatexbase.add_to_callback('define_font', luatextra.define_font, 'luatextra.define_font')
125 end

```

Initialise a few callbacks.

```

126     luatexbase.create_callback('pre_read_file', 'simple', luatextra.default_pre_read)
127     luatexbase.create_callback('file_reader', 'data', luatextra.default_reader)
128     luatexbase.create_callback('file_close', 'simple', luatextra.default_close)
129     luatexbase.add_to_callback('open_read_file', luatextra.open_read_file, 'luatextra.open_read_file')
130     luatexbase.create_callback('font_syntax', 'simple', luatextra.default_font_syntax)
131     luatexbase.create_callback('open_otf_font', 'first', luatextra.default_open_otf)
132     luatexbase.create_callback('post_font_opening', 'simple', luatextra.default_post_font)

```

2 luatextra.sty

2.1 Initializations

First we prevent multiple loads of the file (useful for plain-TeX).

```
133 \csname ifluatextraloaded\endcsname
134 \let\ifluatextraloaded\endinput
135
```

Then we load ifluatex.

```
136
137 \bgroup\expandafter\expandafter\expandafter\egroup
138 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
139 \expandafter\ifx\csname ifluatex\endcsname\relax
140 \input ifluatex.sty
141 \fi
142 \else
143 \RequirePackage{ifluatex}
144 \NeedsTeXFormat{LaTeX2e}
145 \ProvidesPackage{luatextra}
146 [2010/08/02 v0.971 LuaTeX extra low-level macros]
147 \fi
148
```

The two macros `\LuaTeX` and `\LuaLaTeX` are defined to `LuaTeX` and `LuaLaTeX`, because that's the way it's written in the `LuaTeX`'s manual (not in small capitals).

These two macros are the only two loaded if we are under a non-`LuaTeX` engine.

```
149
150 \def\LuaTeX{Lua\TeX }
151 \def\LuaLaTeX{Lua\LaTeX }
152
```

Make sure `LuaTeX` is being used.

```
153 \bgroup\expandafter\expandafter\expandafter\egroup
154 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
155 \input ifluatex.sty
156 \else
157 \RequirePackage{ifluatex}
158 \fi
159 \ifluatex\else
160 \begingroup
161 \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
162 \def\x#1#2{\begingroup\newlinechar10
163 \immediate\write16{Package #1 warning: #2}\endgroup}
164 \else
165 \let\x\PackageWarningNoLine
166 \fi
167 \expandafter\endgroup
```

```

168 \x{luatexbase-modutils}{LuaTeX is required for this package.^^J
169   Aborting package loading.}
170 \expandafter\endinput
171 \fi

    Load other usefull packages.

172 \bgroup\expandafter\expandafter\expandafter\egroup
173 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
174   \input luatexbase-modutils.sty
175   \input luatexbase-attr.sty
176   \input luatexbase-cctb.sty
177   \input luatexbase-regs.sty
178   \input luatexbase-mcb.sty
179 \else
180   \RequirePackage{luatexbase-modutils}
181   \RequirePackage{luatexbase-attr}
182   \RequirePackage{luatexbase-cctb}
183   \RequirePackage{luatexbase-regs}
184   \RequirePackage{luatexbase-mcb}
185 \fi
186 \luatexUseModule{lualibs}

    If the package is loaded with LATEX, we also define the environment luacode.

187 \bgroup\expandafter\expandafter\expandafter\egroup
188 \expandafter\ifx\csname ProvidesPackage\endcsname\relax \else
189   \RequirePackage{environ}
190   \NewEnviron{luacode}{\directlua{\BODY}}
191 \fi

```

2.2 Primitives renaming

Here we differentiate two very different cases: LuaTeX version ≥ 0.36 has no `tex.enableprimitives` function, and has support for multiple lua states, and for versions ≤ 0.35 , the `tex.enableprimitives` is provided, and the old `\directlua` syntax prints a warning.

```

192 \ifnum\luatexversion<36

```

For old versions, we simply rename the primitives. You can note that `\attribute` (and also others) have no `\primitive` before them, because it would make users unable to call `\global\luaattribute`, which is a strong restriction. With this method, we can call it, but if `\attribute` was defined before, this means that `\luaattribute` will get its meaning, which is dangerous. Note also that you cannot use multiple states.

```

193   \def\directlua{\pdfprimitive\directlua0}
194   \def\latelua{\pdfprimitive\latelua0}
195   \def\luadirect{\pdfprimitive\directlua0}
196   \def\lualate{\pdfprimitive\latelua0}
197   \def\luatexattribute{\attribute}
198   \def\luatexattributedef{\attributedef}

```

```

199 \def\luatexclearmarks{\pdfprimitive\luaclearmarks}
200 \def\luatexformatname{\pdfprimitive\formatname}
201 \def\luatexscantexttokens{\pdfprimitive\scantexttokens}
202 \def\luatexcatcodetable{\catcodetable}
203 \def\initluatexcatcodetable{\pdfprimitive\initcatcodetable}
204 \def\saveluatexcatcodetable{\pdfprimitive\savecatcodetable}
205 \def\luaclose{\pdfprimitive\closelua}
206 \else

```

From TeXLive 2009, all primitives should be provided with the `luatex` prefix. For TeXLive 2008, we provide some primitives with this prefix too, to keep backward compatibility.

```

207 \directlua{tex.enableprimitives('luatex', {'attribute'})}
208 \directlua{tex.enableprimitives('luatex', {'attributedef'})}
209 \directlua{tex.enableprimitives('luatex', {'clearmarks'})}
210 \directlua{tex.enableprimitives('luatex', {'formatname'})}
211 \directlua{tex.enableprimitives('luatex', {'scantexttokens'})}
212 \directlua{tex.enableprimitives('luatex', {'catcodetable'})}
213 \directlua{tex.enableprimitives('luatex', {'latelua'})}
214 \directlua{tex.enableprimitives('luatex', {'initcatcodetable'})}
215 \directlua{tex.enableprimitives('luatex', {'savecatcodetable'})}
216 \directlua{tex.enableprimitives('luatex', {'closelua'})}
217 \let\luadirect\directlua
218 \let\lualate\luatexlatelua
219 \let\initluatexcatcodetable\luatexinitcatcodetable
220 \let\saveluatexcatcodetable\luatexsavecatcodetable
221 \let\luaclose\luatexcloselua
222 \fi

```

We load the lua file.

```

223 \directlua{dofile(kpse.find_file("luatextra.lua"))}

```

A small macro to register the `define_font` callback from `luatextra`. See section 1.3 for more details.

```

224 \def\ltxtra@RegisterFontCallback{
225 \directlua{luatextra.register_font_callback()}
226 }

```

We provide some functions for backward compatibility with old versions of `luatextra`.

```

227 \let\newluaattribute\newluatexattribute
228 \let\luaattribute\luatexattribute
229 \let\unsetluaattribute\unsetluatexattribute
230 \let\initluacatcodetable\initluatexcatcodetable
231 \let\luasetcatcoderange\luatexsetcatcoderange
232 \let\newluacatcodetable\newluatexcatcodetable
233 \let\setluaattribute\setluatexattribute
234 \let\luaModuleError\luatexModuleError
235 \let\luaRequireModule\luatexRequireModule
236 \let\luaUseModule\luatexUseModule

```


Finally, we load luaotfload.

```
237 \bgroup\expandafter\expandafter\expandafter\egroup
238 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
239   \input luaotfload.sty
240 \else
241   \RequirePackage{luaotfload}
242 \fi
```