

The `luamplib` package

Hans Hagen, Taco Hoekwater and Elie Roux
`elie.roux@telecom-bretagne.eu`

2009/10/01 v1.02

Abstract

Package to have metapost code typeset directly in a document with Lua_{TEX}.

1 Documentation

This package aims at providing a simple way to typeset directly metapost code in a document with Lua_{TEX}. Lua_{TEX} is built with the lua `mplib` library, that runs metapost code. This package is basically a wrapper (in Lua) for the Lua `mplib` functions and some _{TEX} functions to have the output of the `mplib` functions in the pdf.

The package need to be in PDF mode in order to output something, as PDF specials are not supported by the DVI format and tools.

The metapost figures are put in a _{TEX} `hbox` with dimensions adjusted to the metapost code.

The code is from the `supp-mpl.lua` and `supp-mpl.tex` files from Con_{TEX}t, they have been adapted to L_A_{TEX} and Plain by Elie Roux. The changes are:

- a L_A_{TEX} environment
- all _{TEX} macros start by `mplib`
- use of the `luatextra` printing and module functions
- adapted warning, error and log messages

Using this package is easy: in Plain, type your metapost code between the macros `mplibcode` and `endmpplibcode`, and in L_A_{TEX} in the `mplibcode` environment.

In order to use metapost, some `.mem` files are needed. These files must be generated with the same version of `mplib` as the version of Lua_{TEX}. These files names can be changed, they are by default `mpost.mem` and `mpfun.mem`. If this package is to be included in a distribution, some values may have to be changed in the file `luamplib.lua`, see comments.

If your distribution does not provide valid `.mem` files (TeXLive 2009 will be the first), you'll have to generate and install them by hand, with the script `luamplib-createmem.lua` included in this package if you want them.

These `.mem` files are not mandatory though. If this package doesn't find the `mem` files, it will just input the `.mp` file, and work without problem; the only difference is that it may be a bit slower but this is not noticeable on a modern computer.

2 Files

This package contains three files:

- `luamplib.lua` containing the lua code that calls `mplib`
- `luamplib.sty` containing the macros for L^AT_EX and Plain
- `create-mem.lua`, a standalone `mem` generation script

2.1 `luamplib.lua`

First the `luamplib` module is registered as a LuaT_EX module, with some informations. Here we can't name it `mplib`, as the name is already taken.

```

1
2 luamplib = { }
3
4 luamplib.module = {
5     name      = "luamplib",
6     version   = 1.02,
7     date      = "2009/10/01",
8     description = "Lua package to typeset Metapost with LuaTEX's MPLib.",
9     author    = "Hans Hagen, Taco Hoekwater & Elie Roux",
10    copyright  = "ConTEXt Development Team & Elie Roux",
11    license    = "CC0",
12 }
13
14 luatextra.provides_module(luamplib.module)
15
```

This module is a stripped down version of libraries that are used by ConT_EXt.

```

16
17 local format, concat, abs = string.format, table.concat, math.abs
18
```

The `mem` file and the format name are hardcoded, and they can be set with T_EX if it's useful. The T_EX distributions should change these values if necessary.

```

19
20 luamplib.currentformat = "plain"
21 luamplib.currentmem = "mpost"

```

```

22
23 local currentformat = luamplib.currentformat
24 local currentmem = luamplib.currentmem
25
26 function luamplib.setformat (name)
27     luamplib.currentformat = name
28 end
29
30 function luamplib.setmemfile(name)
31     luamplib.currentmem = name
32 end
33

```

We use the `kpse` library and make it behave like when used with Metapost. To find the `.mem` files with `kpse`, we have to make a small hack... that might be a little bug.

```

34
35 local mpkpse = kpse.new("luatex","mpost")
36
37 function luamplib.finder(name, mode, ftype)
38     if mode == "w" then
39         return name
40     else
41         local result = mpkpse.find_file(name,ftype)
42         if not result and ftype == "mem" then
43             result = mpkpse.find_file("metapost/"..name,ftype)
44         end
45         return result
46     end
47 end
48
49 function luamplib.info (...)
50     luatextra.module_info('luamplib', format(...))
51 end
52
53 function luamplib.log (...)
54     luatextra.module_log('luamplib', format(...))
55 end
56
57 function luamplib.term (...)
58     luatextra.module_term('luamplib', format(...))
59 end
60
61 function luamplib.warning (...)
62     luatextra.module_warning('luamplib', format(...))
63 end
64
65 function luamplib.error (...)
66     luatextra.module_error('luamplib', format(...))
67 end

```

68

This is a small hack for L^AT_EX. In L^AT_EX we read the metapost code line by line, but it needs to be passed entirely to `luamplib.process`, so we simply add the lines in `luamplib.data` and at the end we call `luamplib.process` on `luamplib.data`.

69

```
70 luamplib.data = ""
```

71

```
72 function luamplib.resetdata()
```

```
73     luamplib.data = ""
```

```
74 end
```

75

```
76 function luamplib.addline(line)
```

```
77     luamplib.data = luamplib.data .. '\n' .. line
```

```
78 end
```

79

```
80 function luamplib.processlines()
```

```
81     luamplib.process(luamplib.data)
```

```
82     luamplib.resetdata()
```

```
83 end
```

84

`luamplib.input mp` This function creates a new `mplib` object and inputs the good `.mp` file. It's less efficient than loading the `.mem` file, but it works more safely, as the `.mp` file does not depend on the version of the `mpost` program.

85

```
86 function luamplib.input_mp(name)
```

```
87     local mpx = mplib.new {
```

```
88         ini_version = true,
```

```
89         find_file = luamplib.finder,
```

```
90         job_name = name,
```

```
91     }
```

```
92     mpx:execute(format("input %s ;",name))
```

```
93     return mpx
```

```
94 end
```

95

`luamplib.load` This function is the one loading the metapost format we want. It uses the `luamplib.currentformat` and `luamplib.currentmem` to determine the format and the `mem` file to use.

The rest of this module is not documented. More info can be found in the L^AT_EX manual, articles in user group journals and the files that ship with ConT_EXt.

96

```
97 function luamplib.load()
```

```
98     local mpx = mplib.new {
```

```
99         ini_version = false,
```

```
100         mem_name = currentmem,
```

```

101     find_file = luamplib.finder
102 }
103 if mpx then
104     luamplib.log("using mem file %s", luamplib.finder(currentmem, 'r', 'mem'))
105 else
106     mpx = luamplib.input_mp(currentformat)
107     if mpx then
108         luamplib.log("using mp file %s", luamplib.finder(currentformat, 'r', 'mp'))
109     else
110         luamplib.error("unable to load the metapost format.")
111     end
112 end
113 return mpx
114 end
115
116
117 function luamplib.report(result)
118     if not result then
119         luamplib.error("no result object")
120     elseif result.status > 0 then
121         local t, e, l, f = result.term, result.error, result.log
122         if l then
123             luamplib.log(l)
124         end
125         if t then
126             luamplib.term(t)
127         end
128         if e then
129             if result.status == 1 then
130                 luamplib.warning(e)
131             else
132                 luamplib.error(e)
133             end
134         end
135         if not t and not e and not l then
136             if result.status == 1 then
137                 luamplib.warning("unknown error, no error, terminal or log messages, maybe mis")
138             else
139                 luamplib.error("unknown error, no error, terminal or log messages, maybe mis")
140             end
141         end
142     else
143         return true
144     end
145     return false
146 end
147
148 function luamplib.process(data)
149     local converted, result = false, {}

```

```

150     local mpx = luamplib.load()
151     if mpx and data then
152         local result = mpx:execute(data)
153         if luamplib.report(result) then
154             if result.fig then
155                 converted = luamplib.convert(result)
156             else
157                 luamplib.warning("no figure output")
158             end
159         end
160     else
161         luamplib.error("Mem file unloadable. Maybe generated with a different version of mpl")
162     end
163     return converted, result
164 end
165
166 local function getobjects(result,figure,f)
167     return figure:objects()
168 end
169
170 function luamplib.convert(result, flusher)
171     luamplib.flush(result, flusher)
172     return true -- done
173 end
174
175 local function pdf_startfigure(n,llx,lly,urx,ury)
176     tex.sprint(format("\mplibstarttoPDF{%s}{%s}{%s}{%s}",llx,lly,urx,ury))
177 end
178
179 local function pdf_stopfigure()
180     tex.sprint("\mplibstoptoPDF")
181 end
182
183 function pdf_literalcode(fmt,...) -- table
184     tex.sprint(format("\mplibtoPDF{%s}",format(fmt,...)))
185 end
186
187 function pdf_textfigure(font,size,text,width,height,depth)
188     text = text:gsub(".", "\hbox{%1}") -- kerning happens in metapost
189     tex.sprint(format("\mplibtexttext{%s}{%s}{%s}{%s}{%s}",font,size,text,0,-( 7200/ 7227)/6))
190 end
191
192 local bend_tolerance = 131/65536
193
194 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
195
196 local function pen_characteristics(object)
197     if luamplib.pen_info then
198         local t = luamplib.pen_info(object)
199         rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty

```

```

200         divider = sx*sy - rx*ry
201         return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
202     else
203         rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
204         return false, 1
205     end
206 end
207
208 local function concat(px, py) -- no tx, ty here
209     return (sy*px-ry*py)/divider, (sx*py-rx*px)/divider
210 end
211
212 local function curved(ith,pth)
213     local d = pth.left_x - ith.right_x
214     if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x) <= bend_tolerance then
215         d = pth.left_y - ith.right_y
216         if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y) <= bend_tolerance then
217             return false
218         end
219     end
220     return true
221 end
222
223 local function flushnormalpath(path,open)
224     local pth, ith
225     for i=1,#path do
226         pth = path[i]
227         if not ith then
228             pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
229         elseif curved(ith,pth) then
230             pdf_literalcode("%f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,pth.y_coord)
231         else
232             pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
233         end
234         ith = pth
235     end
236     if not open then
237         local one = path[1]
238         if curved(pth,one) then
239             pdf_literalcode("%f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coord,one.y_coord)
240         else
241             pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
242         end
243     elseif #path == 1 then
244         -- special case .. draw point
245         local one = path[1]
246         pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
247     end
248     return t
249 end

```

```

250
251 local function flushconcatpath(path,open)
252     pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
253     local pth, ith
254     for i=1,#path do
255         pth = path[i]
256         if not ith then
257             pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
258         elseif curved(ith,pth) then
259             local a, b = concat(ith.right_x,ith.right_y)
260             local c, d = concat(pth.left_x,pth.left_y)
261             pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
262         else
263             pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
264         end
265         ith = pth
266     end
267     if not open then
268         local one = path[1]
269         if curved(pth,one) then
270             local a, b = concat(pth.right_x,pth.right_y)
271             local c, d = concat(one.left_x,one.left_y)
272             pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
273         else
274             pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
275         end
276     elseif #path == 1 then
277         -- special case .. draw point
278         local one = path[1]
279         pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
280     end
281     return t
282 end
283

```

Support for specials in DVI has been removed.

```

284
285 function luamplib.flush(result,flusher)
286     if result then
287         local figures = result.fig
288         if figures then
289             for f=1, #figures do
290                 luamplib.log("flushing figure %s",f)
291                 local figure = figures[f]
292                 local objects = getobjects(result,figure,f)
293                 local fignum = tonumber((figure:filename()):match("([%d]+)$")) or figure:char
294                 local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
295                 local bbox = figure:boundingbox()
296                 local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than
297                 if urx < llx then

```



```

298         -- invalid
299         pdf_startfigure(fignum,0,0,0,0)
300         pdf_stopfigure()
301     else
302         pdf_startfigure(fignum,llx,lly,urx,ury)
303         pdf_literalcode("q")
304         if objects then
305             for o=1,#objects do
306                 local object = objects[o]
307                 local objecttype = object.type
308                 if objecttype == "start_bounds" or objecttype == "stop_bounds" then
309                     -- skip
310                 elseif objecttype == "start_clip" then
311                     pdf_literalcode("q")
312                     flushnormalpath(object.path,t,false)
313                     pdf_literalcode("W n")
314                 elseif objecttype == "stop_clip" then
315                     pdf_literalcode("Q")
316                     miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
317                 elseif objecttype == "special" then
318                     -- not supported
319                 elseif objecttype == "text" then
320                     local ot = object.transform -- 3,4,5,6,1,2
321                     pdf_literalcode("q %f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
322                     pdf_textfigure(object.font,object.dsize,object.text,object.w)
323                     pdf_literalcode("Q")
324                 else
325                     local cs = object.color
326                     if cs and #cs > 0 then
327                         pdf_literalcode(luamplib.colorconverter(cs))
328                     end
329                     local ml = object.miterlimit
330                     if ml and ml ~= miterlimit then
331                         miterlimit = ml
332                         pdf_literalcode("%f M",ml)
333                     end
334                     local lj = object.linejoin
335                     if lj and lj ~= linejoin then
336                         linejoin = lj
337                         pdf_literalcode("%i j",lj)
338                     end
339                     local lc = object.linecap
340                     if lc and lc ~= linecap then
341                         linecap = lc
342                         pdf_literalcode("%i J",lc)
343                     end
344                     local dl = object.dash
345                     if dl then
346                         local d = format("[%s] %i d",concat(dl.dashes or {}, " "))
347                         if d ~= dashed then

```

```

348         dashed = d
349         pdf_literalcode(dashed)
350     end
351 elseif dashed then
352     pdf_literalcode("[] 0 d")
353     dashed = false
354 end
355 local path = object.path
356 local transformed, penwidth = false, 1
357 local open = path and path[1].left_type and path[#path].right_type
358 local pen = object.pen
359 if pen then
360     if pen.type == 'elliptical' then
361         transformed, penwidth = pen_characteristics(object)
362         pdf_literalcode("%f w", penwidth)
363         if objecttype == 'fill' then
364             objecttype = 'both'
365         end
366     else -- calculated by mplib itself
367         objecttype = 'fill'
368     end
369 end
370 if transformed then
371     pdf_literalcode("q")
372 end
373 if path then
374     if transformed then
375         flushconcatpath(path, open)
376     else
377         flushnormalpath(path, open)
378     end
379     if objecttype == "fill" then
380         pdf_literalcode("h f")
381     elseif objecttype == "outline" then
382         pdf_literalcode((open and "S") or "h S")
383     elseif objecttype == "both" then
384         pdf_literalcode("h B")
385     end
386 end
387 if transformed then
388     pdf_literalcode("Q")
389 end
390 local path = object.htap
391 if path then
392     if transformed then
393         pdf_literalcode("q")
394     end
395     if transformed then
396         flushconcatpath(path, open)
397     else

```

```

398             flushnormalpath(path,open)
399         end
400         if objecttype == "fill" then
401             pdf_literalcode("h f")
402         elseif objecttype == "outline" then
403             pdf_literalcode((open and "S") or "h S")
404         elseif objecttype == "both" then
405             pdf_literalcode("h B")
406         end
407         if transformed then
408             pdf_literalcode("Q")
409         end
410     end
411     if cr then
412         pdf_literalcode(cr)
413     end
414 end
415 end
416 end
417 pdf_literalcode("Q")
418 pdf_stopfigure()
419 end
420 end
421 end
422 end
423 end
424
425 function luamplib.colorconverter(cr)
426     local n = #cr
427     if n == 4 then
428         local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
429         return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
430     elseif n == 3 then
431         local r, g, b = cr[1], cr[2], cr[3]
432         return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
433     else
434         local s = cr[1]
435         return format("%.3f g %.3f G",s,s), "0 g 0 G"
436     end
437 end

```

2.2 luamplib.sty

First we need to load fancyvrb, to define the environment mplibcode.

```

438
439 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
440 \input luatextra.sty
441 \else
442 \NeedsTeXFormat{LaTeX2e}

```

```

443 \ProvidesPackage{luamplib}
444 [2009/10/01 v1.02 mplib package for LuaTeX.]
445 \RequirePackage{luatextra}
446 \RequirePackage{fancyvrb}
447 \fi
448

```

Loading of lua code.

```

449
450 \luatexUseModule{luamplib}
451

```

There are (basically) two formats for metapost: *plain* and *mpfun*. The corresponding .mem files are (at least will be) `luatex-plain.mem` and `luatex-mpfun.mem` in T_EXLive. With these functions you can set the format and the mem files that will be used by this package. Warning: the package never generates the mem files, you have to do it by hand, with `create-mem.lua`.

```

452
453 \def\mplibsetformat#1{\directlua0{luamplib.setformat([[#1]])}}
454
455 \def\mplibsetmemfile#1{\directlua0{luamplib.setmemfile([[#1]])}}
456

```

MPLib only works in PDF mode, we don't do anything if we are in DVI mode, and we output a warning.

```

457
458 \ifnum\pdfoutput>0
459   \let\mplibtoPDF\pdfliteral
460 \else
461   %\def\MPLIBtoPDF#1{\special{pdf:literal direct #1}} % not ok yet
462   \def\mplibtoPDF#1{}
463   \expandafter\ifx\csname PackageWarning\endcsname\relax
464     \write16{}
465     \write16{Warning: MPLib only works in PDF mode, no figure will be output.}
466     \write16{}
467   \else
468     \PackageWarning{mplib}{MPLib only works in PDF mode, no figure will be output.}
469   \fi
470 \fi
471

```

The Plain-specific stuff.

```

472
473 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
474
475 \def\mplibsetupcatcodes{
476   \catcode'\{=12 \catcode'\}=12 \catcode'\#=12 \catcode'\^=12 \catcode'\~=12
477   \catcode'\_ =12 \catcode'\%=12 \catcode'\&=12 \catcode'\$=12
478 }
479

```

```

480 \def\mplibcode{%
481   \bgroup %
482   \mplibsetupcatcodes %
483   \mplibdocode %
484 }
485
486 \long\def\mplibdocode#1\endmplibcode{%
487   \egroup %
488   \mplibprocess{#1}%
489 }
490
491 \long\def\mplibprocess#1{%
492   \luadirect{luamplib.process([[#1]])}%
493 }
494
495 \else
496

```

The L^AT_EX-specific parts. First a Hack for the catcodes in L^AT_EX.

```

497
498 \makeatletter
499 \begingroup
500 \catcode'\,=13
501 \catcode'\-=13
502 \gdef\FV@hack{%
503   \def,{\string,}%
504   \def-{\string-}%
505 }
506 \endgroup
507

```

In L^AT_EX (it's not the case in plainT_EX), we get the metapost code line by line, here is the function handling a line.

```

508
509 \newcommand\mplibaddlines[1]{%
510   \begingroup %
511   \FV@hack %
512   \def\FV@ProcessLine##1{%
513     \luadirect{luamplib.addline([[##1]])}%
514   }%
515   \csname FV@SV@#1\endcsname %
516   \endgroup %
517 }
518
519 \makeatother
520

```

The L^AT_EX environment is a modified `verbatim` environment.

```

521
522 \newenvironment{mplibcode}{%
523   \VerbatimEnvironment %

```

```

524 \begin{SaveVerbatim}{memoire}%
525 }{%
526 \end{SaveVerbatim}%
527 \mplibaddlines{memoire}%
528 \luadirect{luamplib.processlines()}%
529 }
530
531 \fi
532
    We use a dedicated scratchbox.

533
534 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
535
    We encapsulate the literals.

536
537 \def\mplibstarttoPDF#1#2#3#4{
538 \hbox\bgroup
539 \xdef\MPllx{#1}\xdef\MPlly{#2}%
540 \xdef\MPurx{#3}\xdef\MPury{#4}%
541 \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
542 \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
543 \parskip0pt%
544 \leftskip0pt%
545 \parindent0pt%
546 \everypar{}%
547 \setbox\mplibscratchbox\vbox\bgroup
548 \noindent
549 }
550
551 \def\mplibstoptoPDF{%
552 \egroup %
553 \setbox\mplibscratchbox\hbox %
554 {\hskip-\MPllx bp%
555 \raise-\MPlly bp%
556 \box\mplibscratchbox}%
557 \setbox\mplibscratchbox\vbox to \MPheight
558 {\vfill
559 \hsize\MPwidth
560 \wd\mplibscratchbox0pt%
561 \ht\mplibscratchbox0pt%
562 \dp\mplibscratchbox0pt%
563 \box\mplibscratchbox}%
564 \wd\mplibscratchbox\MPwidth
565 \ht\mplibscratchbox\MPheight
566 \box\mplibscratchbox
567 \egroup
568 }
569

```

Text items have a special handler.

```

570
571 \def\mplibtexttext#1#2#3#4#5{%
572   \begingroup
573   \setbox\mplibscratchbox\hbox
574     {\font\temp=#1 at #2bp%
575     \temp
576     #3}%
577   \setbox\mplibscratchbox\hbox
578     {\hskip#4 bp%
579     \raise#5 bp%
580     \box\mplibscratchbox}%
581   \wd\mplibscratchbox0pt%
582   \ht\mplibscratchbox0pt%
583   \dp\mplibscratchbox0pt%
584   \box\mplibscratchbox
585   \endgroup
586 }
587

```

2.3 luamplib-createmem.lua

Finally a small standalone file to call with `texlua` that generates `luatex-plain.mem` in the current directory. To generate other formats in other names, simply change the last line. After the `mem` generation, you'll have to install it in a directory searchable by `TEX`.

```

588
589 kpse.set_program_name("kpsewhich")
590
591 function finder (name, mode, ftype)
592   if mode == "w" then
593     return name
594   else
595     local result = kpse.find_file(name,ftype)
596     return result
597   end
598 end
599
600 local preamble = [[
601 input %s ; dump ;
602 ]]
603

```

`makeformat`

```

604
605 makeformat = function (name, mem_name)
606   local mpx = mplib.new {
607     ini_version = true,

```

```

608         find_file = finder,
609         job_name = mem_name,
610     }
611     if mpx then
612         local result
613         result = mpx:execute(string.format(preamble,name))
614         print(string.format("dumping format %s in %s", name, mem_name))
615         mpx:finish()
616     end
617 end
618
619 makeformat("plain", "luatex-plain.mem")
620

```