

The `luamplib` package

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

2009/09/23 v1.01

Abstract

Package to have metapost code typeset directly in a document with LuaTeX.

1 Documentation

This package aims at providing a simple way to typeset directly metapost code in a document with LuaTeX. LuaTeX 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-mp1.lua` and `supp-mp1.tex` files from ConTeXt, they have been adapted to L^ATeX and Plain by Elie Roux. The changes are:

- a L^ATeX environment
- all TeX macros start by `mplib`
- use of the `luatextra` printing and module functions

Using this package is easy: in Plain, type your metapost code between the macros `mplibcode` and `endmplibcode`, and in L^ATeX 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 LuaTeX. These files names can be changed, they are by default `luatex-plain.mem` and `luatex-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.

2 Files

This package contains three files:

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

2.1 `luamplib.lua`

First the `luamplib` module is registered as a \LaTeX 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   = 1.01,
7   date      = "2009/09/23",
8   description = "Lua functions to typeset Metapost directly with 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 ConTeXt .

```
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 \TeX if it's useful. The \TeX distributions should change these values if necessary.

```
19
20 luamplib.currentformat = "plain"
21 luamplib.currentmem = "luatex-plain.mem"
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 unless a finder is already defined. To find the `.mem` files with `kpse`, we have to set the `engine` environment variable to `metapost`.

```

34
35 luamplib.finder = luamplib.finder or function(name, mode, ftype)
36   if mode == "w" then
37     return name
38   else
39     local result
40     if ftype == 'mem' then
41       local envsave = os.getenv('engine')
42       os.setenv('engine', 'metapost')
43       result = kpse.find_file(name,ftype)
44       os.setenv('engine', envsave)
45     else
46       result = kpse.find_file(name,ftype)
47     end
48     return result
49   end
50 end
51
52 function luamplib.info (...)
53   luatextra.module_info('luamplib', format(...))
54 end
55
56 function luamplib.log (...)
57   luatextra.module_log('luamplib', format(...))
58 end
59
60 function luamplib.term (...)
61   luatextra.module_term('luamplib', format(...))
62 end
63
64 function luamplib.warning (...)
65   luatextra.module_warning('luamplib', format(...))
66 end
67
68 function luamplib.error (...)
69   luatextra.module_error('luamplib', format(...))
70 end
71

```

This is a small hack for \LaTeX . In \LaTeX 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`.

```

72
73 luamplib.data = ""
74

```

```

75 function luamplib.resetdata()
76     luamplib.data = ""
77 end
78
79 function luamplib.addline(line)
80     luamplib.data = luamplib.data .. '\n' .. line
81 end
82
83 function luamplib.processlines()
84     luamplib.process(luamplib.data)
85     luamplib.resetdata()
86 end
87
88
89

```

`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 LuaTeX manual, articles in user group journals and the files that ship with ConTeXt.

```

90
91 function luamplib.load()
92     local mpx = mplib.new {
93         ini_version = false,
94         mem_name = currentmem,
95         find_file = luamplib.finder
96     }
97     if mpx then
98         luamplib.log("using mem file %s", luamplib.finder(currentmem, 'r', 'mem'))
99         return mpx, nil
100     else
101         return nil, { status = 99, error = "out of memory or invalid format" }
102     end
103 end
104
105
106 function luamplib.report(result)
107     if not result then
108         luamplib.error("no result object")
109     elseif result.status > 0 then
110         local t, e, l, f = result.term, result.error, result.log
111         if l then
112             luamplib.log(l)
113         end
114         if t then
115             luamplib.term(t)

```

```

116         end
117         if e then
118             if result.status == 1 then
119                 luamplib.warning(e)
120             else
121                 luamplib.error(e)
122             end
123         end
124         if not t and not e and not l then
125             if result.status == 1 then
126                 luamplib.warning("unknown error, no error, terminal or log messages, maybe mis")
127             else
128                 luamplib.error("unknown error, no error, terminal or log messages, maybe mis")
129             end
130         end
131     else
132         return true
133     end
134     return false
135 end
136
137 function luamplib.process(data)
138     local converted, result = false, {}
139     local mpx = luamplib.load()
140     if mpx and data then
141         local result = mpx:execute(data)
142         if luamplib.report(result) then
143             if result.fig then
144                 converted = luamplib.convert(result)
145             else
146                 luamplib.warning("no figure output")
147             end
148         end
149     else
150         luamplib.error("Mem file unloadable. Maybe generated with a different version of mpl")
151     end
152     return converted, result
153 end
154
155 local function getobjects(result,figure,f)
156     return figure:objects()
157 end
158
159 function luamplib.convert(result, flusher)
160     luamplib.flush(result, flusher)
161     return true -- done
162 end
163
164 local function pdf_startfigure(n,llx,lly,urx,ury)
165     tex.sprint(format("\mplibstarttoPDF{%s}{%s}{%s}{%s}",llx,lly,urx,ury))

```

```

166 end
167
168 local function pdf_stopfigure()
169     tex.sprint("\mplibstoptoPDF")
170 end
171
172 function pdf_literalcode(fmt,...) -- table
173     tex.sprint(format("\mplibtoPDF{%s}",format(fmt,...)))
174 end
175
176 function pdf_textfigure(font,size,text,width,height,depth)
177     text = text:gsub(".", "\\hbox{%1}") -- kerning happens in metapost
178     tex.sprint(format("\mplibtexttext{%s}{%s}{%s}{%s}{%s}",font,size,text,0,-( 7200/ 7227)/6
179 end
180
181 local bend_tolerance = 131/65536
182
183 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
184
185 local function pen_characteristics(object)
186     if luamplib.pen_info then
187         local t = luamplib.pen_info(object)
188         rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
189         divider = sx*sy - rx*ry
190         return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
191     else
192         rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
193         return false, 1
194     end
195 end
196
197 local function concat(px, py) -- no tx, ty here
198     return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
199 end
200
201 local function curved(ith,pth)
202     local d = pth.left_x - ith.right_x
203     if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x
204         d = pth.left_y - ith.right_y
205         if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.le
206             return false
207         end
208     end
209     return true
210 end
211
212 local function flushnormalpath(path,open)
213     local pth, ith
214     for i=1,#path do
215         pth = path[i]

```

```

216         if not ith then
217             pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
218         elseif curved(ith,pth) then
219             pdf_literalcode("%f %f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y)
220         else
221             pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
222         end
223         ith = pth
224     end
225     if not open then
226         local one = path[1]
227         if curved(pth,one) then
228             pdf_literalcode("%f %f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y)
229         else
230             pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
231         end
232     elseif #path == 1 then
233         -- special case .. draw point
234         local one = path[1]
235         pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
236     end
237     return t
238 end
239
240 local function flushconcatpath(path,open)
241     pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
242     local pth, ith
243     for i=1,#path do
244         pth = path[i]
245         if not ith then
246             pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
247         elseif curved(ith,pth) then
248             local a, b = concat(ith.right_x,ith.right_y)
249             local c, d = concat(pth.left_x,pth.left_y)
250             pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
251         else
252             pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
253         end
254         ith = pth
255     end
256     if not open then
257         local one = path[1]
258         if curved(pth,one) then
259             local a, b = concat(pth.right_x,pth.right_y)
260             local c, d = concat(one.left_x,one.left_y)
261             pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
262         else
263             pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
264         end
265     elseif #path == 1 then

```

```

266         -- special case .. draw point
267         local one = path[1]
268         pdf_literalcode("%f %f 1",concat(one.x_coord,one.y_coord))
269     end
270     return t
271 end
272

```

Support for specials in DVI has been removed.

```

273
274 function luamplib.flush(result,flusher)
275     if result then
276         local figures = result.fig
277         if figures then
278             for f=1, #figures do
279                 luamplib.info("flushing figure %s",f)
280                 local figure = figures[f]
281                 local objects = getobjects(result,figure,f)
282                 local fignum = tonumber((figure:filename()):match("([%d]+)$") or figure:char
283                 local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
284                 local bbox = figure:boundingbox()
285                 local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than
286                 if urx < llx then
287                     -- invalid
288                     pdf_startfigure(fignum,0,0,0,0)
289                     pdf_stopfigure()
290                 else
291                     pdf_startfigure(fignum,llx,lly,urx,ury)
292                     pdf_literalcode("q")
293                     if objects then
294                         for o=1,#objects do
295                             local object = objects[o]
296                             local objecttype = object.type
297                             if objecttype == "start_bounds" or objecttype == "stop_bounds" t
298                                 -- skip
299                             elseif objecttype == "start_clip" then
300                                 pdf_literalcode("q")
301                                 flushnormalpath(object.path,t,false)
302                                 pdf_literalcode("W n")
303                             elseif objecttype == "stop_clip" then
304                                 pdf_literalcode("Q")
305                                 miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
306                             elseif objecttype == "special" then
307                                 -- not supported
308                             elseif objecttype == "text" then
309                                 local ot = object.transform -- 3,4,5,6,1,2
310                                 pdf_literalcode("q %f %f %f %f %f %f cm",ot[3],ot[4],ot[5],o
311                                 pdf_textfigure(object.font,object.dsize,object.text,object.w
312                                 pdf_literalcode("Q")
313                             else

```



```

314     local cs = object.color
315     if cs and #cs > 0 then
316         pdf_literalcode(luamplib.colorconverter(cs))
317     end
318     local ml = object.miterlimit
319     if ml and ml ~= miterlimit then
320         miterlimit = ml
321         pdf_literalcode("%f M",ml)
322     end
323     local lj = object.linejoin
324     if lj and lj ~= linejoin then
325         linejoin = lj
326         pdf_literalcode("%i j",lj)
327     end
328     local lc = object.linecap
329     if lc and lc ~= linecap then
330         linecap = lc
331         pdf_literalcode("%i J",lc)
332     end
333     local dl = object.dash
334     if dl then
335         local d = format("[%s] %i d",concat(dl.dashes or {}, " "))
336         if d ~= dashed then
337             dashed = d
338             pdf_literalcode(dashed)
339         end
340     elseif dashed then
341         pdf_literalcode("[] 0 d")
342         dashed = false
343     end
344     local path = object.path
345     local transformed, penwidth = false, 1
346     local open = path and path[1].left_type and path[#path].right_type
347     local pen = object.pen
348     if pen then
349         if pen.type == 'elliptical' then
350             transformed, penwidth = pen_characteristics(object)
351             pdf_literalcode("%f w",penwidth)
352             if objecttype == 'fill' then
353                 objecttype = 'both'
354             end
355         else -- calculated by mplib itself
356             objecttype = 'fill'
357         end
358     end
359     if transformed then
360         pdf_literalcode("q")
361     end
362     if path then
363         if transformed then

```

```

364         flushconcatpath(path,open)
365     else
366         flushnormalpath(path,open)
367     end
368     if objecttype == "fill" then
369         pdf_literalcode("h f")
370     elseif objecttype == "outline" then
371         pdf_literalcode((open and "S") or "h S")
372     elseif objecttype == "both" then
373         pdf_literalcode("h B")
374     end
375 end
376 if transformed then
377     pdf_literalcode("Q")
378 end
379 local path = object.htap
380 if path then
381     if transformed then
382         pdf_literalcode("q")
383     end
384     if transformed then
385         flushconcatpath(path,open)
386     else
387         flushnormalpath(path,open)
388     end
389     if objecttype == "fill" then
390         pdf_literalcode("h f")
391     elseif objecttype == "outline" then
392         pdf_literalcode((open and "S") or "h S")
393     elseif objecttype == "both" then
394         pdf_literalcode("h B")
395     end
396     if transformed then
397         pdf_literalcode("Q")
398     end
399 end
400 if cr then
401     pdf_literalcode(cr)
402 end
403 end
404 end
405 end
406 pdf_literalcode("Q")
407 pdf_stopfigure()
408 end
409 end
410 end
411 end
412 end
413

```

```

414 function luamplib.colorconverter(cr)
415     local n = #cr
416     if n == 4 then
417         local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
418         return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
419     elseif n == 3 then
420         local r, g, b = cr[1], cr[2], cr[3]
421         return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
422     else
423         local s = cr[1]
424         return format("%.3f g %.3f G",s,s), "0 g 0 G"
425     end
426 end

```

2.2 luamplib.sty

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

```

427
428 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
429     \input luatextra.sty
430 \else
431     \NeedsTeXFormat{LaTeX2e}
432     \ProvidesPackage{luamplib}
433     [2009/09/23 v1.01 mplib package for LuaTeX.]
434     \RequirePackage{luatextra}
435     \RequirePackage{fancyvrb}
436 \fi
437

```

Loading of lua code.

```

438
439 \luaTeXUseModule{luamplib}
440

```

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`.

```

441
442 \def\mplibsetformat#1{\directlua0{luamplib.setformat([[#1]])}}
443
444 \def\mplibsetmemfile#1{\directlua0{luamplib.setmemfile([[#1]])}}
445

```

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

```

446
447 \ifnum\pdfoutput>0

```

```

448 \let\mplibtoPDF\pdfliteral
449 \else
450 %\def\MPLIBtoPDF#1{\special{pdf:literal direct #1}} % not ok yet
451 \def\mplibtoPDF#1{}
452 \expandafter\ifx\csname PackageWarning\endcsname\relax
453 \write16{}
454 \write16{Warning: MPLib only works in PDF mode, no figure will be output.}
455 \write16{}
456 \else
457 \PackageWarning{mplib}{MPLib only works in PDF mode, no figure will be output.}
458 \fi
459 \fi
460

```

The Plain-specific stuff.

```

461
462 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
463
464 \def\mplibsetupcatcodes{
465 \catcode'\{=12 \catcode'\}=12 \catcode'\#=12 \catcode'\^=12 \catcode'\~=12
466 \catcode'\_ =12 \catcode'\%=12 \catcode'\&=12 \catcode'\$=12
467 }
468
469 \def\mplibcode{%
470 \bgroup %
471 \mplibsetupcatcodes %
472 \mplibdocode %
473 }
474
475 \long\def\mplibdocode#1\endmplibcode{%
476 \egroup %
477 \mplibprocess{#1}%
478 }
479
480 \long\def\mplibprocess#1{%
481 \luadirect{luamplib.process([[#1]])}%
482 }
483
484 \else
485

```

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

```

486
487 \makeatletter
488 \begingroup
489 \catcode'\,=13
490 \catcode'\-=13
491 \gdef\FV@hack{%
492 \def,{\string,}%
493 \def-{\string-}%

```

```

494 }
495 \endgroup
496

```

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.

```

497
498 \newcommand\mplibaddlines[1]{%
499   \begingroup %
500   \FV@hack %
501   \def\FV@ProcessLine##1{%
502     \luadirect{luamplib.addline([[##1]])}%
503   }%
504   \csname FV@SV@#1\endcsname %
505   \endgroup %
506 }
507
508 \makeatother
509

```

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

```

510
511 \newenvironment{mplibcode}{%
512   \VerbatimEnvironment %
513   \begin{SaveVerbatim}{memoire}%
514 }{%
515   \end{SaveVerbatim}%
516   \mplibaddlines{memoire}%
517   \luadirect{luamplib.processlines()}%
518 }
519
520 \fi
521

```

We use a dedicated scratchbox.

```

522
523 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
524

```

We encapsulate the literals.

```

525
526 \def\mplibstarttoPDF#1#2#3#4{
527   \hbox\bgroup
528   \xdef\MPllx{#1}\xdef\MPlly{#2}%
529   \xdef\MPurx{#3}\xdef\MPury{#4}%
530   \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
531   \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
532   \parskip0pt%
533   \leftskip0pt%
534   \parindent0pt%
535   \everypar{}%

```

```

536 \setbox\mplibscratchbox\vbox\bgroup
537 \noindent
538 }
539
540 \def\mplibstoptoPDF{%
541 \egroup %
542 \setbox\mplibscratchbox\hbox %
543 {\hskip-\MPllx bp%
544 \raise-\MPlly bp%
545 \box\mplibscratchbox}%
546 \setbox\mplibscratchbox\vbox to \MPheight
547 {\vfill
548 \hsize\MPwidth
549 \wd\mplibscratchboxOpt%
550 \ht\mplibscratchboxOpt%
551 \dp\mplibscratchboxOpt%
552 \box\mplibscratchbox}%
553 \wd\mplibscratchbox\MPwidth
554 \ht\mplibscratchbox\MPheight
555 \box\mplibscratchbox
556 \egroup
557 }
558

```

Text items have a special handler.

```

559
560 \def\mplibtexttext#1#2#3#4#5{%
561 \begingroup
562 \setbox\mplibscratchbox\hbox
563 {\font\temp=#1 at #2bp%
564 \temp
565 #3}%
566 \setbox\mplibscratchbox\hbox
567 {\hskip#4 bp%
568 \raise#5 bp%
569 \box\mplibscratchbox}%
570 \wd\mplibscratchboxOpt%
571 \ht\mplibscratchboxOpt%
572 \dp\mplibscratchboxOpt%
573 \box\mplibscratchbox
574 \endgroup
575 }
576

```

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 T_EX.

```
577
578 kpse.set_program_name("kpsewhich")
579
580 function finder (name, mode, ftype)
581     if mode == "w" then
582         return name
583     else
584         local result = kpse.find_file(name,ftype)
585         return result
586     end
587 end
588
589 local preamble = [[
590 input %s ; dump ;
591 ]]
592
makeformat
593
594 makeformat = function (name, mem_name)
595     local mpx = mplib.new {
596         ini_version = true,
597         find_file = finder,
598         job_name = mem_name,
599     }
600     if mpx then
601         local result
602         result = mpx:execute(string.format(preamble,name))
603         print(string.format("dumping format %s in %s", name, mem_name))
604         mpx:finish()
605     end
606 end
607
608 makeformat("plain", "luatex-plain.mem")
609
```