The **luamplib** package

Hans Hagen, Taco Hoekwater and Elie Roux
Maintainer: Manuel Pégourié-Gonnard — Support: <lualatex-dev@tug.org>

2011/12/09 v1.09

Abstract

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

1 Documentation

This packages 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 needs 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 ConTeXt, they have been adapted to LaTeX and Plain by Elie Roux. The changes are:

- a LaTeX environment

- all TeX macros start by `mplib`

- use of luatexbase for errors, warnings and declaration

Using this package is easy: in Plain, type your metapost code between the macros `mplibcode` and `endmplibcode`, and in LaTeX in the `mplibcode` environment.

There are (basically) two formats for metapost: *plain* and *mpfun*. By default, the *plain* format is used, but you can set the format to be used by future figures at any time using `\mplibsetformat{⟨format name⟩}`.

2 Implementation

2.1 Lua module

1 ⟨*lua⟩

Use the `luamplib` namespace, since `mplib` is for the metapost library itself.
2 `module('luamplib', package.seeall)`

Identification, and additional reporting function, for compatibility with the original `report()` function and associated data structure.
3 `local err, warn, info, log = luatexbase.provides_module({`

```
 4     name          = "luamplib",
 5     version       =  1.09,
 6     date          = "2011/11/09",
 7     description   = "Lua package to typeset Metapost with LuaTeX's MPLib.",
 8 })
 9
10 local function term(...)
11     texio.write_nl('term', 'luamplib: ' .. string.format(...))
12 end
13
```

This module is a stripped down version of libraries that are used by ConTEXt. Provide a few "shortcuts" expected by the imported code.

```
14 local format, concat, abs = string.format, table.concat, math.abs
15
```

This is a small trick for LATEX. In LATEX we read the metapost code line by line, but it needs to be passed entirely to `process()`, so we simply add the lines in `data` and at the end we call `process(data)`.

```
16 local data = ""
17
18 local function resetdata()
19     data = ""
20 end
21 function addline(line)
22     data = data .. '\n' .. line
23 end
24 function processlines()
25     process(data)
26     resetdata()
27 end
28
```

As the finder function for `mplib`, use the `kpse` library and make it behave like as if MetaPost was used (or almost, since the engine name is not set this way—not sure if this is a problem).

```
29 local mpkpse = kpse.new("luatex", "mpost")
30
31 local function finder(name, mode, ftype)
32     if mode == "w" then
33         return name
34     else
35         return mpkpse:find_file(name,ftype)
36     end
37 end
38
```

Default format name, and a public function to change it.

```
39 local currentformat = "plain"
40
41 function setformat (name)
42     currentformat = name
43 end
44
```

Create a new mplib object and input the correct .mp file as given by `currentformat`. With older versions of MetaPost, using a `mem` file may be more efficient, but newer versions don't

support it any more so for the sake of simplicity, don't even try to use a `mem` file. `ini_version` is ignored by new versions but is useful to prevent old ones from trying to load a `mem` file.

```
45 function load()
46     local mpx = mplib.new {
47         find_file   = finder,
48         ini_version = true,
49     }
50     mpx:execute(format("input %s ;", currentformat))
51     return mpx
52 end
53
```

The rest of this module is not documented. More information can be found in the LuaTeX manual, articles in user group journals and the files that ship with ConTeXt.

```
54 function report(result)
55     if not result then
56         err("no result object")
57     elseif result.status > 0 then
58         local t, e, l, f = result.term, result.error, result.log
59         if l then
60             log(l)
61         end
62         if t then
63             term(t)
64         end
65         if e then
66             if result.status == 1 then
67                 warn(e)
68             else
69                 err(e)
70             end
71         end
72         if not t and not e and not l then
73             if result.status == 1 then
74                 warn("unknown error, no error, terminal or log messages, maybe missing beginfig/endfig")
75             else
76                 err("unknown error, no error, terminal or log messages, maybe missing beginfig/endfig")
77             end
78         end
79     else
80         return true
81     end
82     return false
83 end
84
85 function process(data)
86     local converted, result = false, {}
87     local mpx = load()
88     if mpx and data then
89         local result = mpx:execute(data)
90         if report(result) then
91             if result.fig then
92                 converted = convert(result)
93             else
```

```
 94                warn("no figure output")
 95            end
 96        end
 97    else
 98        err("Mem file unloadable. Maybe generated with a different version of mplib?")
 99    end
100    return converted, result
101 end
102
103 local function getobjects(result,figure,f)
104    return figure:objects()
105 end
106
107 function convert(result, flusher)
108    flush(result, flusher)
109    return true -- done
110 end
111
112 local function pdf_startfigure(n,llx,lly,urx,ury)
113    tex.sprint(format("\\mplibstarttoPDF{%s}{%s}{%s}{%s}",llx,lly,urx,ury))
114 end
115
116 local function pdf_stopfigure()
117    tex.sprint("\\mplibstoptoPDF")
118 end
119
120 function pdf_literalcode(fmt,...) -- table
121    tex.sprint(format("\\mplibtoPDF{%s}",format(fmt,...)))
122 end
123
124 function pdf_textfigure(font,size,text,width,height,depth)
125    text = text:gsub(".","\\hbox{%1}") -- kerning happens in metapost
126    tex.sprint(format("\\mplibtextext{%s}{%s}{%s}{%s}{%s}",font,size,text,0,-( 7200/ 7227)/65536*depth))
127 end
128
129 local bend_tolerance = 131/65536
130
131 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
132
133 local function pen_characteristics(object)
134    if mplib.pen_info then
135        local t = mplib.pen_info(object)
136        rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
137        divider = sx*sy - rx*ry
138        return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
139    else
140        rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
141        return false, 1
142    end
143 end
144
145 local function concat(px, py) -- no tx, ty here
146    return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
```

4

```lua
147 end
148
149 local function curved(ith,pth)
150     local d = pth.left_x - ith.right_x
151     if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x - d) <= bend
152         d = pth.left_y - ith.right_y
153         if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y - d) <= b
154             return false
155         end
156     end
157     return true
158 end
159
160 local function flushnormalpath(path,open)
161     local pth, ith
162     for i=1,#path do
163         pth = path[i]
164         if not ith then
165             pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
166         elseif curved(ith,pth) then
167             pdf_literalcode("%f %f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coo
168         else
169             pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
170         end
171         ith = pth
172     end
173     if not open then
174         local one = path[1]
175         if curved(pth,one) then
176             pdf_literalcode("%f %f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coo
177         else
178             pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
179         end
180     elseif #path == 1 then
181         -- special case .. draw point
182         local one = path[1]
183         pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
184     end
185     return t
186 end
187
188 local function flushconcatpath(path,open)
189     pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
190     local pth, ith
191     for i=1,#path do
192         pth = path[i]
193         if not ith then
194             pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
195         elseif curved(ith,pth) then
196             local a, b = concat(ith.right_x,ith.right_y)
197             local c, d = concat(pth.left_x,pth.left_y)
198             pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
199         else
```

5

```
200        pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
201     end
202     ith = pth
203  end
204  if not open then
205     local one = path[1]
206     if curved(pth,one) then
207        local a, b = concat(pth.right_x,pth.right_y)
208        local c, d = concat(one.left_x,one.left_y)
209        pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
210     else
211        pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
212     end
213  elseif #path == 1 then
214     -- special case .. draw point
215     local one = path[1]
216     pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
217  end
218  return t
219 end
220
```

Support for specials in DVI has been removed.
```
221 function flush(result,flusher)
222    if result then
223       local figures = result.fig
224       if figures then
225          for f=1, #figures do
226             log("flushing figure %s",f)
227             local figure = figures[f]
228             local objects = getobjects(result,figure,f)
229             local fignum = tonumber((figure:filename()):match("([%d]+)$") or figure:charcode() or 0)
230             local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
231             local bbox = figure:boundingbox()
232             local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
233             if urx < llx then
234                -- invalid
235                pdf_startfigure(fignum,0,0,0,0)
236                pdf_stopfigure()
237             else
238                pdf_startfigure(fignum,llx,lly,urx,ury)
239                pdf_literalcode("q")
240                if objects then
241                   for o=1,#objects do
242                      local object = objects[o]
243                      local objecttype = object.type
244                      if objecttype == "start_bounds" or objecttype == "stop_bounds" then
245                         -- skip
246                      elseif objecttype == "start_clip" then
247                         pdf_literalcode("q")
248                         flushnormalpath(object.path,t,false)
249                         pdf_literalcode("W n")
250                      elseif objecttype == "stop_clip" then
251                         pdf_literalcode("Q")
```

```lua
252                            miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
253                        elseif objecttype == "special" then
254                            -- not supported
255                        elseif objecttype == "text" then
256                            local ot = object.transform -- 3,4,5,6,1,2
257                            pdf_literalcode("q %f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot
258                            pdf_textfigure(object.font,object.dsize,object.text,object.width,object.h
259                            pdf_literalcode("Q")
260                        else
261                            local cs = object.color
262                            if cs and #cs > 0 then
263                                pdf_literalcode(colorconverter(cs))
264                            end
265                            local ml = object.miterlimit
266                            if ml and ml ~= miterlimit then
267                                miterlimit = ml
268                                pdf_literalcode("%f M",ml)
269                            end
270                            local lj = object.linejoin
271                            if lj and lj ~= linejoin then
272                                linejoin = lj
273                                pdf_literalcode("%i j",lj)
274                            end
275                            local lc = object.linecap
276                            if lc and lc ~= linecap then
277                                linecap = lc
278                                pdf_literalcode("%i J",lc)
279                            end
280                            local dl = object.dash
281                            if dl then
282                                local d = format("[%s] %i d",concat(dl.dashes or {}," "),dl.offset)
283                                if d ~= dashed then
284                                    dashed = d
285                                    pdf_literalcode(dashed)
286                                end
287                            elseif dashed then
288                                pdf_literalcode("[] 0 d")
289                                dashed = false
290                            end
291                            local path = object.path
292                            local transformed, penwidth = false, 1
293                            local open = path and path[1].left_type and path[#path].right_type
294                            local pen = object.pen
295                            if pen then
296                                if pen.type == 'elliptical' then
297                                    transformed, penwidth = pen_characteristics(object) -- boolean, va
298                                    pdf_literalcode("%f w",penwidth)
299                                    if objecttype == 'fill' then
300                                        objecttype = 'both'
301                                    end
302                                else -- calculated by mplib itself
303                                    objecttype = 'fill'
304                                end
```

7

```
305                                        end
306                                        if transformed then
307                                            pdf_literalcode("q")
308                                        end
309                                        if path then
310                                            if transformed then
311                                                flushconcatpath(path,open)
312                                            else
313                                                flushnormalpath(path,open)
314                                            end
315                                            if objecttype == "fill" then
316                                                pdf_literalcode("h f")
317                                            elseif objecttype == "outline" then
318                                                pdf_literalcode((open and "S") or "h S")
319                                            elseif objecttype == "both" then
320                                                pdf_literalcode("h B")
321                                            end
322                                        end
323                                        if transformed then
324                                            pdf_literalcode("Q")
325                                        end
326                                        local path = object.htap
327                                        if path then
328                                            if transformed then
329                                                pdf_literalcode("q")
330                                            end
331                                            if transformed then
332                                                flushconcatpath(path,open)
333                                            else
334                                                flushnormalpath(path,open)
335                                            end
336                                            if objecttype == "fill" then
337                                                pdf_literalcode("h f")
338                                            elseif objecttype == "outline" then
339                                                pdf_literalcode((open and "S") or "h S")
340                                            elseif objecttype == "both" then
341                                                pdf_literalcode("h B")
342                                            end
343                                            if transformed then
344                                                pdf_literalcode("Q")
345                                            end
346                                        end
347                                        if cr then
348                                            pdf_literalcode(cr)
349                                        end
350                                    end
351                                end
352                            end
353                    pdf_literalcode("Q")
354                    pdf_stopfigure()
355                end
356            end
357        end
```

```
358        end
359 end
360
361 function colorconverter(cr)
362     local n = #cr
363     if n == 4 then
364         local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
365         return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
366     elseif n == 3 then
367         local r, g, b = cr[1], cr[2], cr[3]
368         return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
369     else
370         local s = cr[1]
371         return format("%.3f g %.3f G",s,s), "0 g 0 G"
372     end
373 end
374 ⟨/lua⟩
```

2.2   TEX package

```
375 ⟨∗package⟩
```

First we need to load fancyvrb, to define the environment mplibcode.
```
376 \bgroup\expandafter\expandafter\expandafter\egroup
377 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
378   \input luatexbase-modutils.sty
379 \else
380   \NeedsTeXFormat{LaTeX2e}
381   \ProvidesPackage{luamplib}
382     [2011/12/09 v1.09 mplib package for LuaTeX]
383   \RequirePackage{luatexbase-modutils}
384   \RequirePackage{fancyvrb}
385 \fi
```

Loading of lua code.
```
386 \RequireLuaModule{luamplib}
```

Set the format for metapost.
```
387 \def\mplibsetformat#1{%
388   \directlua{luamplib.setformat("\luatexluaescapestring{#1}")}}
```

MPLib only works in PDF mode, we don't do anything if we are in DVI mode, and we output a warning.
```
389 \ifnum\pdfoutput>0
390     \let\mplibtoPDF\pdfliteral
391 \else
392     %\def\MPLIBtoPDF#1{\special{pdf:literal direct #1}} % not ok yet
393     \def\mplibtoPDF#1{}
394     \expandafter\ifx\csname PackageWarning\endcsname\relax
395       \write16{}
396       \write16{Warning: MPLib only works in PDF mode, no figure will be output.}
397       \write16{}
398     \else
399       \PackageWarning{mplib}{MPLib only works in PDF mode, no figure will be output.}
400     \fi
401 \fi
```

9

The Plain-specific stuff.

```
402 \bgroup\expandafter\expandafter\expandafter\egroup
403 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
404 \def\mplibsetupcatcodes{%
405   \catcode`\{=12 \catcode`\}=12 \catcode`\#=12 \catcode`\^=12 \catcode`\~=12
406   \catcode`\_=12 \catcode`\%=12 \catcode`\&=12 \catcode`\$=12
407 }
408 \def\mplibcode{%
409   \bgroup %
410   \mplibsetupcatcodes %
411   \mplibdocode %
412 }
413 \long\def\mplibdocode#1\endmplibcode{%
414   \egroup %
415   \mplibprocess{#1}%
416 }
417 \long\def\mplibprocess#1{%
418   \directlua{luamplib.process("\luatexluaescapestring{#1}")}%
419 }
420 \else
```

The LaTeX-specific parts. First a Hack for the catcodes in LaTeX.

```
421 \begingroup
422 \catcode`\,=13
423 \catcode`\-=13
424 \catcode`\<=13
425 \catcode`\>=13
426 \catcode`\^^I=13
427 \catcode`\`=13 % must be last...
428 \gdef\FV@hack{%
429   \def,{\string,}%
430   \def-{\string-}%
431   \def<{\string<}%
432   \def>{\string>}%
433   \def`{\string`}%
434   \def^^I{\string^^I}%
435 }
436 \endgroup
```

In LaTeX (it's not the case in plainTeX), we get the metapost code line by line, here is the function handling a line.

```
437 \newcommand\mplibaddlines[1]{%
438   \begingroup %
439   \FV@hack %
440   \def\FV@ProcessLine##1{%
441     \directlua{luamplib.addline("\luatexluaescapestring{##1}")}%
442   }%
443   \csname FV@SV@#1\endcsname %
444   \endgroup %
445 }
```

The LaTeX environment is a modified `verbatim` environment.

```
446 \newenvironment{mplibcode}{%
447   \VerbatimEnvironment %
448   \begin{SaveVerbatim}{memoire}%
```

```
449 }{%
450   \end{SaveVerbatim}%
451   \mplibaddlines{memoire}%
452   \directlua{luamplib.processlines()}%
453 }
454
455 \fi
```

We use a dedicated scratchbox.

```
456 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
```

We encapsulate the litterals.

```
457 \def\mplibstarttoPDF#1#2#3#4{%
458   \hbox\bgroup
459   \xdef\MPllx{#1}\xdef\MPlly{#2}%
460   \xdef\MPurx{#3}\xdef\MPury{#4}%
461   \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
462   \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
463   \parskip0pt%
464   \leftskip0pt%
465   \parindent0pt%
466   \everypar{}%
467   \setbox\mplibscratchbox\vbox\bgroup
468   \noindent
469 }
470 \def\mplibstoptoPDF{%
471   \egroup %
472   \setbox\mplibscratchbox\hbox %
473     {\hskip-\MPllx bp%
474     \raise-\MPlly bp%
475     \box\mplibscratchbox}%
476   \setbox\mplibscratchbox\vbox to \MPheight
477     {\vfill
478     \hsize\MPwidth
479     \wd\mplibscratchbox0pt%
480     \ht\mplibscratchbox0pt%
481     \dp\mplibscratchbox0pt%
482     \box\mplibscratchbox}%
483   \wd\mplibscratchbox\MPwidth
484   \ht\mplibscratchbox\MPheight
485   \box\mplibscratchbox
486   \egroup
487 }
```

Text items have a special handler.

```
488 \def\mplibtextext#1#2#3#4#5{%
489   \begingroup
490   \setbox\mplibscratchbox\hbox
491     {\font\temp=#1 at #2bp%
492     \temp
493     #3}%
494   \setbox\mplibscratchbox\hbox
495     {\hskip#4 bp%
496     \raise#5 bp%
497     \box\mplibscratchbox}%
```

```
498   \wd\mplibscratchbox0pt%
499   \ht\mplibscratchbox0pt%
500   \dp\mplibscratchbox0pt%
501   \box\mplibscratchbox
502   \endgroup
503 }
```

That's all folks!

```
504 ⟨/package⟩
```