The luamplib package

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

2011/06/23 v1.08

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       =  1.08,
6     date          = "2011/06/23",
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.

```
function load()
    local mpx = mplib.new {
        find_file   = finder,
        ini_version = true,
    }
    mpx:execute(format("input %s ;", currentformat))
    return mpx
end

```

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.

```
function report(result)
    if not result then
        err("no result object")
    elseif result.status > 0 then
        local t, e, l, f = result.term, result.error, result.log
        if l then
            log(l)
        end
        if t then
            term(t)
        end
        if e then
            if result.status == 1 then
                warn(e)
            else
                err(e)
            end
        end
        if not t and not e and not l then
            if result.status == 1 then
                warn("unknown error, no error, terminal or log messages, maybe missing beginfig/endfig")
            else
                err("unknown error, no error, terminal or log messages, maybe missing beginfig/endfig")
            end
        end
    else
        return true
    end
    return false
end

function process(data)
    local converted, result = false, {}
    local mpx = load()
    if mpx and data then
        local result = mpx:execute(data)
        if report(result) then
            if result.fig then
                converted = convert(result)
            else
```

```
                warn("no figure output")
            end
        end
    else
        err("Mem file unloadable. Maybe generated with a different version of mplib?")
    end
    return converted, result
end

local function getobjects(result,figure,f)
    return figure:objects()
end

function convert(result, flusher)
    flush(result, flusher)
    return true -- done
end

local function pdf_startfigure(n,llx,lly,urx,ury)
    tex.sprint(format("\\mplibstarttoPDF{%s}{%s}{%s}{%s}",llx,lly,urx,ury))
end

local function pdf_stopfigure()
    tex.sprint("\\mplibstoptoPDF")
end

function pdf_literalcode(fmt,...) -- table
    tex.sprint(format("\\mplibtoPDF{%s}",format(fmt,...)))
end

function pdf_textfigure(font,size,text,width,height,depth)
    text = text:gsub(".","\\hbox{%1}") -- kerning happens in metapost
    tex.sprint(format("\\mplibtextext{%s}{%s}{%s}{%s}{%s}",font,size,text,0,-( 7200/ 7227)/65536*depth))
end

local bend_tolerance = 131/65536

local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1

local function pen_characteristics(object)
    if mplib.pen_info then
        local t = mplib.pen_info(object)
        rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
        divider = sx*sy - rx*ry
        return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
    else
        rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
        return false, 1
    end
end

local function concat(px, py) -- no tx, ty here
    return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
```

```
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
```

```
            pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
        end
        ith = pth
    end
    if not open then
        local one = path[1]
        if curved(pth,one) then
            local a, b = concat(pth.right_x,pth.right_y)
            local c, d = concat(one.left_x,one.left_y)
            pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
        else
            pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
        end
    elseif #path == 1 then
        -- special case .. draw point
        local one = path[1]
        pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
    end
    return t
end
```

Support for specials in DVI has been removed.

```
function flush(result,flusher)
    if result then
        local figures = result.fig
        if figures then
            for f=1, #figures do
                log("flushing figure %s",f)
                local figure = figures[f]
                local objects = getobjects(result,figure,f)
                local fignum = tonumber((figure:filename()):match("([%d]+)$") or figure:charcode() or 0)
                local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
                local bbox = figure:boundingbox()
                local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
                if urx < llx then
                    -- invalid
                    pdf_startfigure(fignum,0,0,0,0)
                    pdf_stopfigure()
                else
                    pdf_startfigure(fignum,llx,lly,urx,ury)
                    pdf_literalcode("q")
                    if objects then
                        for o=1,#objects do
                            local object = objects[o]
                            local objecttype = object.type
                            if objecttype == "start_bounds" or objecttype == "stop_bounds" then
                                -- skip
                            elseif objecttype == "start_clip" then
                                pdf_literalcode("q")
                                flushnormalpath(object.path,t,false)
                                pdf_literalcode("W n")
                            elseif objecttype == "stop_clip" then
                                pdf_literalcode("Q")
```

```
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
```

```
                                end
                                if transformed then
                                    pdf_literalcode("q")
                                end
                                if path then
                                    if transformed then
                                        flushconcatpath(path,open)
                                    else
                                        flushnormalpath(path,open)
                                    end
                                    if objecttype == "fill" then
                                        pdf_literalcode("h f")
                                    elseif objecttype == "outline" then
                                        pdf_literalcode((open and "S") or "h S")
                                    elseif objecttype == "both" then
                                        pdf_literalcode("h B")
                                    end
                                end
                                if transformed then
                                    pdf_literalcode("Q")
                                end
                                local path = object.htap
                                if path then
                                    if transformed then
                                        pdf_literalcode("q")
                                    end
                                    if transformed then
                                        flushconcatpath(path,open)
                                    else
                                        flushnormalpath(path,open)
                                    end
                                    if objecttype == "fill" then
                                        pdf_literalcode("h f")
                                    elseif objecttype == "outline" then
                                        pdf_literalcode((open and "S") or "h S")
                                    elseif objecttype == "both" then
                                        pdf_literalcode("h B")
                                    end
                                    if transformed then
                                        pdf_literalcode("Q")
                                    end
                                end
                                if cr then
                                    pdf_literalcode(cr)
                                end
                            end
                        end
                    end
                    pdf_literalcode("Q")
                    pdf_stopfigure()
                end
            end
        end
```

```
    end
end

function colorconverter(cr)
    local n = #cr
    if n == 4 then
        local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
        return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
    elseif n == 3 then
        local r, g, b = cr[1], cr[2], cr[3]
        return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
    else
        local s = cr[1]
        return format("%.3f g %.3f G",s,s), "0 g 0 G"
    end
end
```

⟨/lua⟩

2.2 TEX package

⟨∗package⟩

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

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname ProvidesPackage\endcsname\relax
  \input luatexbase-modutils.sty
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luamplib}
    [2011/06/23 v1.08 mplib package for LuaTeX]
  \RequirePackage{luatexbase-modutils}
  \RequirePackage{fancyvrb}
\fi
```

Loading of lua code.

```
\RequireLuaModule{luamplib}
```

Set the format for metapost.

```
\def\mplibsetformat#1{%
  \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.

```
\ifnum\pdfoutput>0
    \let\mplibtoPDF\pdfliteral
\else
    %\def\MPLIBtoPDF#1{\special{pdf:literal direct #1}} % not ok yet
    \def\mplibtoPDF#1{}
    \expandafter\ifx\csname PackageWarning\endcsname\relax
      \write16{}
      \write16{Warning: MPLib only works in PDF mode, no figure will be output.}
      \write16{}
    \else
      \PackageWarning{mplib}{MPLib only works in PDF mode, no figure will be output.}
    \fi
\fi
```

The Plain-specific stuff.

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname ProvidesPackage\endcsname\relax
\def\mplibsetupcatcodes{
  \catcode`\{=12 \catcode`\}=12 \catcode`\#=12 \catcode`\^=12 \catcode`\~=12
  \catcode`\_=12 \catcode`\%=12 \catcode`\&=12 \catcode`\$=12
}
\def\mplibcode{%
  \bgroup %
  \mplibsetupcatcodes %
  \mplibdocode %
}
\long\def\mplibdocode#1\endmplibcode{%
  \egroup %
  \mplibprocess{#1}%
}
\long\def\mplibprocess#1{%
  \directlua{luamplib.process("\luatexluaescapestring{#1}")}%
}
\else
```

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

```
\begingroup
\catcode`\,=13
\catcode`\-=13
\catcode`\<=13
\catcode`\>=13
\catcode`\^^I=13
\catcode`\`=13 % must be last...
\gdef\FV@hack{%
  \def,{\string,}%
  \def-{\string-}%
  \def<{\string<}%
  \def>{\string>}%
  \def`{\string`}%
  \def^^I{\string^^I}%
}
\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.

```
\newcommand\mplibaddlines[1]{%
  \begingroup %
  \FV@hack %
  \def\FV@ProcessLine##1{%
    \directlua{luamplib.addline("\luatexluaescapestring{##1}")}%
  }%
  \csname FV@SV@#1\endcsname %
  \endgroup %
}
```

The LaTeX environment is a modified `verbatim` environment.

```
\newenvironment{mplibcode}{%
  \VerbatimEnvironment %
  \begin{SaveVerbatim}{memoire}%
```

```
}{%
  \end{SaveVerbatim}%
  \mplibaddlines{memoire}%
  \directlua{luamplib.processlines()}%
}

\fi
```

We use a dedicated scratchbox.

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

We encapsulate the litterals.

```
\def\mplibstarttoPDF#1#2#3#4{
  \hbox\bgroup
  \xdef\MPllx{#1}\xdef\MPlly{#2}%
  \xdef\MPurx{#3}\xdef\MPury{#4}%
  \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
  \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
  \parskip0pt%
  \leftskip0pt%
  \parindent0pt%
  \everypar{}%
  \setbox\mplibscratchbox\vbox\bgroup
  \noindent
}
\def\mplibstoptoPDF{%
  \egroup %
  \setbox\mplibscratchbox\hbox %
    {\hskip-\MPllx bp%
     \raise-\MPlly bp%
     \box\mplibscratchbox}%
  \setbox\mplibscratchbox\vbox to \MPheight
    {\vfill
     \hsize\MPwidth
     \wd\mplibscratchbox0pt%
     \ht\mplibscratchbox0pt%
     \dp\mplibscratchbox0pt%
     \box\mplibscratchbox}%
  \wd\mplibscratchbox\MPwidth
  \ht\mplibscratchbox\MPheight
  \box\mplibscratchbox
  \egroup
}
```

Text items have a special handler.

```
\def\mplibtextext#1#2#3#4#5{%
  \begingroup
  \setbox\mplibscratchbox\hbox
    {\font\temp=#1 at #2bp%
     \temp
     #3}%
  \setbox\mplibscratchbox\hbox
    {\hskip#4 bp%
     \raise#5 bp%
     \box\mplibscratchbox}%
```

```
  \wd\mplibscratchbox0pt%
  \ht\mplibscratchbox0pt%
  \dp\mplibscratchbox0pt%
  \box\mplibscratchbox
  \endgroup
}
```

That's all folks!

⟨/package⟩