The luamplib package

Hans Hagen, Taco Hoekwater, Elie Roux, Philipp Gesang and Kim Dohyun
Maintainer: LuaLaTeX Maintainers — Support: <lualatex-dev@tug.org>

2018/01/04 v2.12.2

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.

In the past, the package required PDF mode in order to output something. Starting with version 2.7 it works in DVI mode as well, though DVIPDFMx is the only DVI tool currently supported.

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

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

The code is from the `luatex-mplib.lua` and `luatex-mplib.tex` files from ConTEXt, they have been adapted to LATEX and Plain by Elie Roux and Philipp Gesang, new functionalities have been added by Kim Dohyun. The changes are:

- a LATEX environment
- all TEX macros start by `mplib`
- use of luatexbase for errors, warnings and declaration
- possibility to use `btex ... etex` to typeset TEX code. `textext()` is a more versatile macro equivalent to `TEX()` from TEX.mp. `TEX()` is also allowed and is a synomym of `textext()`.

  N.B. Since v2.5, `btex ... etex` input from external `mp` files will also be processed by luamplib. However, `verbatimtex ... etex` will be entirely ignored in this case.

- `verbatimtex ... etex` (in TeX file) that comes just before `beginfig()` is not ignored, but the TeX code inbetween will be inserted before the following mplib hbox. Using this command, each mplib box can be freely moved horizontally and/or vertically. Also, a box number might be assigned to mplib box, allowing it to be reused later (see test files). E.G.

  ```
  \mplibcode
  verbatimtex \moveright 3cm etex; beginfig(0); ... endfig;
  verbatimtex \leavevmode etex; beginfig(1); ... endfig;
  verbatimtex \leavevmode\lower 1ex etex; beginfig(2); ... endfig;
  verbatimtex \endgraf\moveright 1cm etex; beginfig(3); ... endfig;
  \endmplibcode
  ```

  N.B. `\endgraf` should be used instead of `\par` inside `verbatimtex ... etex`.

- TeX code in `VerbatimTeX(...)` or `verbatimtex ... etex` (in TeX file) between `beginfig()` and `endfig` will be inserted after flushing out the mplib figure. E.G.

  ```
  \mplibcode
    D := sqrt(2)**7;
    beginfig(0);
    draw fullcircle scaled D;
    VerbatimTeX("\gdef\Dia{" & decimal D & "}");
    endfig;
  \endmplibcode
  diameter: \Dia bp.
  ```

- Notice that, after each figure is processed, macro `\MPwidth` stores the width value of latest figure; `\MPheight`, the height value. Incidentally, also note that `\MPllx`, `\MPlly`, `\MPurx`, and `\MPury` store the bounding box information of latest figure without the unit `bp`.

- Since v2.3, new macros `\everymplib` and `\everyendmplib` redefine token lists `\everymplibtoks` and `\everyendmplibtoks` respectively, which will be automatically inserted at the beginning and ending of each mplib code. E.G.

  ```
  \everymplib{ verbatimtex \leavevmode etex; beginfig(0); }
  \everyendmplib{ endfig; }
  \mplibcode % beginfig/endfig not needed; always in horizontal mode
    draw fullcircle scaled 1cm;
  \endmplibcode
  ```

  N.B. Many users have complained that mplib figures do not respect alignment commands such as `\centering` or `\raggedleft`. That's because luamplib does not force horizontal or vertical mode. If you want all mplib figures center- (or right-) aligned, please use `\everymplib` command with `\leavevmode` as shown above.

- Since v2.3, `\mpdim` and other raw TEX commands are allowed inside mplib code. This feature is inpired by gmp.sty authored by Enrico Gregorio. Please refer the manual of gmp package for details. E.G.

  ```
  \begin{mplibcode}
    draw origin--(\mpdim{\linewidth},0) withpen pencircle scaled 4
    dashed evenly scaled 4 withcolor \mpcolor{orange};
  \end{mplibcode}
  ```

  N.B. Users should not use the protected variant of `btex ... etex` as provided by gmp package. As **luamplib** automatically protects TEX code inbetween, `\btex` is not supported here.

- With `\mpcolor` command, color names or expressions of **color**/**xcolor** packages can be used inside mplibcode enviroment, though **luamplib** does not automatically load these packages. See the example code above. For spot colors, **(x)spotcolor** (in PDF mode) and **xespotcolor** (in DVI mode) packages are supported as well.

- Users can choose `numbersystem` option since v2.4. The default value `scaled` can be changed to `double` by declaring `\mplibnumbersystem{double}`. For details see `http://github.com/lualatex/luamplib/issues/21`.

- To support `btex ... etex` in external `.mp` files, **luamplib** inspects the content of each and every `.mp` input files and makes caches if nececcsary, before returning their paths to LuaTEX's mplib library. This would make the compilation time longer wastefully, as most `.mp` files do not contain `btex ... etex` command. So **luamplib** provides macros as follows, so that users can give instruction about files that do not require this functionality.

  – `\mplibmakenocache{<filename>[,<filename>,...]}`

  – `\mplibcancelnocache{<filename>[,<filename>,...]}`

  where `<filename>` is a file name excluding `.mp` extension. Note that `.mp` files under `$TEXMFMAIN/metapost/base` and `$TEXMFMAIN/metapost/context/base` are already registered by default.

- By default, cache files will be stored in `$TEXMFVAR/luamplib_cache` or, if it's not available, in the same directory as where pdf/dvi output file is saved. This however can be changed by the command `\mplibcachedir{<directory path>}`, where tilde (`~`) is interpreted as the user's home directory (on a windows machine as well). As backslashes (`\`) should be escaped by users, it would be easier to use slashes (`/`) instead.

- Starting with v2.6, `\mplibtextextlabel{enable}` enables string labels typeset via `textext()` instead of `infont` operator. So, `label("my text",origin)` thereafter is exactly the same as `label(textext("my text"),origin)`. N.B. In the background, **luamplib** redefines `infont` operator so that the right side argument (the font part) is totally ignored. Every string label therefore will be typeset with current TEX font. Also take care of `char` operator in the left side argument, as this might bring unpermitted characters into TEX.

- Starting with v2.9, `\mplibcodeinherit{enable}` enables the inheritance of variables, constants, and macros defined by previous `mplibcode` chunks. On the contrary, the default value `\mplibcodeinherit{disable}` will make each code chunks being treated as an independent instance, and never affected by previous code chunks.

  N.B. To inherit `btex ... etex` labels as well as metapost variables, it is necessary to declare `\mplibglobaltextext{enable}` in advance. On this case, be careful that normal TEX boxes can conflict with `btex ... etex` boxes, though this would occur very rarely. Notwithstanding the danger, it is a 'must' option to activate `\mplibglobaltextext` if you want to use `graph.mp` with `\mplibcodeinherit` functionality.

  ```
  \mplibcodeinherit{enable}
  \mplibglobaltextext{enable}
  \everymplib{ beginfig(0);} \everyendmplib{ endfig;}
  \mplibcode
    label(btex $\sqrt{2}$ etex, origin);
    draw fullcircle scaled 20;
    picture pic; pic := currentpicture;
  \endmplibcode
  \mplibcode
    currentpicture := pic scaled 2;
  \endmplibcode
  ```

- Starting with v2.11, users can issue `\mplibverbatim{enable}`, after which the contents of mplibcode environment will be read verbatim. As a result, users cannot use `\mpdim`, `\mpcolor` etc. All TEX commands outside of `btex ... etex` or `verbatimtex ... etex` are not expanded and will be fed literally into the mplib process.

- At the end of package loading, **luamplib** searches `luamplib.cfg` and, if found, reads the file in automatically. Frequently used settings such as `\everymplib` or `\mplibcachedir` are suitable for going into this file.

There are (basically) two formats for metapost: *plain* and *metafun*. 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

Use the `luamplib` namespace, since `mplib` is for the metapost library itself. ConTEXt uses `metapost`.

```

luamplib          = luamplib or { }

```

Identification.

```

local luamplib    = luamplib
luamplib.showlog  = luamplib.showlog or false
luamplib.lastlog  = ""

luatexbase.provides_module {
  name          = "luamplib",
  version       = "2.12.2",
  date          = "2018/01/04",
  description   = "Lua package to typeset Metapost with LuaTeX's MPLib.",
}

```

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

```

local format, abs = string.format, math.abs

local err  = function(...) return luatexbase.module_error  ("luamplib", format(...)) end
local warn = function(...) return luatexbase.module_warning("luamplib", format(...)) end
local info = function(...) return luatexbase.module_info   ("luamplib", format(...)) end

local stringgsub    = string.gsub
local stringfind    = string.find
local stringmatch   = string.match
local stringgmatch  = string.gmatch
local stringexplode = string.explode
local tableconcat   = table.concat
local texsprint     = tex.sprint
local textprint     = tex.tprint

local texget      = tex.get
local texgettoks  = tex.gettoks
local texgetbox   = tex.getbox

local mplib = require ('mplib')
local kpse  = require ('kpse')
local lfs   = require ('lfs')

local lfsattributes = lfs.attributes
local lfsisdir      = lfs.isdir
local lfsmkdir      = lfs.mkdir
local lfstouch      = lfs.touch
local ioopen        = io.open

local file = file or { }
```

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)`.

A few helpers, taken from `l-file.lua`.

```
local replacesuffix = file.replacesuffix or function(filename, suffix)
  return (stringgsub(filename,"%.[%a%d]+$","")) .. "." .. suffix
end
local stripsuffix = file.stripsuffix or function(filename)
  return (stringgsub(filename,"%.[%a%d]+$",""))
end

```

`btex ... etex` in input .mp files will be replaced in finder.

```
local is_writable = file.is_writable or function(name)
  if lfsisdir(name) then
    name = name .. "/_luam_plib_temp_file_"
    local fh = ioopen(name,"w")
    if fh then
      fh:close(); os.remove(name)
      return true
    end
  end
end
local mk_full_path = lfs.mkdirs or function(path)
  local full = ""
  for sub in stringgmatch(path,"(/*[^\\/]+)") do
    full = full .. sub
    lfsmkdir(full)
  end
end

local luamplibtime = kpse.find_file("luamplib.lua")
luamplibtime = luamplibtime and lfsattributes(luamplibtime,"modification")

local currenttime = os.time()

local outputdir
if lfstouch then
  local texmfvar = kpse.expand_var('$TEXMFVAR')
  if texmfvar and texmfvar ~= "" and texmfvar ~= '$TEXMFVAR' then
    for _,dir in next,stringexplode(texmfvar,os.type == "windows" and ";" or ":") do
      if not lfsisdir(dir) then
        mk_full_path(dir)
      end
      if is_writable(dir) then
        local cached = format("%s/luamplib_cache",dir)
        lfsmkdir(cached)
        outputdir = cached
        break
      end
    end
  end
end
if not outputdir then
  outputdir = "."
  for _,v in ipairs(arg) do
    local t = stringmatch(v,"%-output%-directory=(.+)")
    if t then
      outputdir = t
      break
    end
  end
end

function luamplib.getcachedir(dir)
  dir = dir:gsub("##","#")
  dir = dir:gsub("^~",
    os.type == "windows" and os.getenv("UserProfile") or os.getenv("HOME"))
  if lfstouch and dir then
    if lfsisdir(dir) then
      if is_writable(dir) then
        luamplib.cachedir = dir
      else
        warn("Directory '"..dir.."' is not writable!")
      end
    else
      warn("Directory '"..dir.."' does not exist!")
    end
  end
end

local noneedtoreplace = {
  ["boxes.mp"] = true,
  --  ["format.mp"] = true,
  ["graph.mp"] = true,
  ["marith.mp"] = true,
  ["mfplain.mp"] = true,
  ["mpost.mp"] = true,
  ["plain.mp"] = true,
  ["rboxes.mp"] = true,
  ["sarith.mp"] = true,
  ["string.mp"] = true,
  ["TEX.mp"] = true,
  ["metafun.mp"] = true,
  ["metafun.mpiv"] = true,
  ["mp-abck.mpiv"] = true,
  ["mp-apos.mpiv"] = true,
  ["mp-asnc.mpiv"] = true,
  ["mp-bare.mpiv"] = true,
  ["mp-base.mpiv"] = true,
  ["mp-butt.mpiv"] = true,
  ["mp-char.mpiv"] = true,
  ["mp-chem.mpiv"] = true,
  ["mp-core.mpiv"] = true,
  ["mp-crop.mpiv"] = true,
  ["mp-figs.mpiv"] = true,
  ["mp-form.mpiv"] = true,
  ["mp-func.mpiv"] = true,
  ["mp-grap.mpiv"] = true,
  ["mp-grid.mpiv"] = true,
  ["mp-grph.mpiv"] = true,
  ["mp-idea.mpiv"] = true,
  ["mp-luas.mpiv"] = true,
  ["mp-mlib.mpiv"] = true,
  ["mp-page.mpiv"] = true,
  ["mp-shap.mpiv"] = true,
  ["mp-step.mpiv"] = true,
  ["mp-text.mpiv"] = true,
  ["mp-tool.mpiv"] = true,
}
luamplib.noneedtoreplace = noneedtoreplace

local function replaceformatmp(file,newfile,ofmodify)
  local fh = ioopen(file,"r")
  if not fh then return file end
  local data = fh:read("*all"); fh:close()
  fh = ioopen(newfile,"w")
  if not fh then return file end
  fh:write(
    "let normalinfont = infont;\n",
    "primarydef str infont name = rawtextext(str) enddef;\n",
    data,
    "vardef Fmant_(expr x) = rawtextext(decimal abs x) enddef;\n",
    "vardef Fexp_(expr x) = rawtextext(\"$^{\"&decimal x&\"}$\") enddef;\n",
    "let infont = normalinfont;\n"
  ); fh:close()
  lfstouch(newfile,currenttime,ofmodify)
  return newfile
end

local esctex  = "!!!T!!!E!!!X!!!"
local esclbr  = "!!!!!LEFTBRCE!!!!!"
local escrbr  = "!!!!!RGHTBRCE!!!!!"
local escpcnt = "!!!!!PERCENT!!!!!"
local eschash = "!!!!!HASH!!!!!"
local begname = "%f[A-Z_a-z]"
local endname = "%f[^A-Z_a-z]"

local btex_etex        = begname.."btex"..endname.."%s*(.-)%s*"..begname.."etex"..endname
local verbatimtex_etex = begname.."verbatimtex"..endname.."%s*(.-)%s*"..begname.."etex"..endname

local function protecttexcontents(str)
  return str:gsub("\\%%", "\\"..escpcnt)
            :gsub("%%.-\n", "")
            :gsub("%%.-$",  "")
            :gsub('"', '"&ditto&"')
            :gsub("\n%s*", " ")
            :gsub(escpcnt, "%%")
end

local function replaceinputmpfile (name,file)
  local ofmodify = lfsattributes(file,"modification")
  if not ofmodify then return file end
  local cachedir = luamplib.cachedir or outputdir
  local newfile = name:gsub("%W","_")
  newfile = cachedir .."/luamplib_input_"..newfile
  if newfile and luamplibtime then
    local nf = lfsattributes(newfile)
    if nf and nf.mode == "file" and ofmodify == nf.modification and luamplibtime < nf.access then
      return nf.size == 0 and file or newfile
    end
  end
  if name == "format.mp" then return replaceformatmp(file,newfile,ofmodify) end

  local fh = ioopen(file,"r")
  if not fh then return file end
  local data = fh:read("*all"); fh:close()

  local count,cnt = 0,0

  data = data:gsub("\"[^\n]-\"", function(str)
    return str:gsub("([bem])tex"..endname,"%1"..esctex)
  end)

  data, cnt = data:gsub(btex_etex, function(str)
    return format("rawtextext(\"%s\")",protecttexcontents(str))
  end)
  count = count + cnt
  data, cnt = data:gsub(verbatimtex_etex, "")
  count = count + cnt

  data = data:gsub("\"[^\n]-\"", function(str) -- restore string btex .. etex
    return str:gsub("([bem])"..esctex, "%1tex")
  end)

  if count == 0 then
    noneedtoreplace[name] = true
    fh = ioopen(newfile,"w");
    if fh then
      fh:close()
      lfstouch(newfile,currenttime,ofmodify)
    end
    return file
  end
  fh = ioopen(newfile,"w")
  if not fh then return file end
  fh:write(data); fh:close()
  lfstouch(newfile,currenttime,ofmodify)
  return newfile
end

local randomseed = nil
```

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).

```

local mpkpse = kpse.new("luatex", "mpost")

local special_ftype = {
  pfb = "type1 fonts",
  enc = "enc files",
}

local function finder(name, mode, ftype)
  if mode == "w" then
    return name
  else
    ftype = special_ftype[ftype] or ftype
    local file = mpkpse:find_file(name,ftype)
    if file then
      if not lfstouch or ftype ~= "mp" or noneedtoreplace[name] then
        return file
      end
      return replaceinputmpfile(name,file)
    end
    return mpkpse:find_file(name,stringmatch(name,"[a-zA-Z]+$"))
  end
end
luamplib.finder = finder

```

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.

```lua

function luamplib.resetlastlog()
  luamplib.lastlog = ””
end

```

Below included is section that defines fallbacks for older versions of mplib.

```lua
local mplibone = tonumber(mplib.version()) <= 1.50

if mplibone then

  luamplib.make = luamplib.make or function(name,mem_name,dump)
    local t = os.clock()
    local mpx = mplib.new {
      ini_version = true,
      find_file = luamplib.finder,
      job_name = stripsuffix(name)
    }
    mpx:execute(format(”input %s ;”,name))
    if dump then
      mpx:execute(”dump ;”)
      info(”format %s made and dumped for %s in %0.3f seconds”,mem_name,name,os.clock()-t)
    else
      info(”%s read in %0.3f seconds”,name,os.clock()-t)
    end
    return mpx
  end

  function luamplib.load(name)
    local mem_name = replacesuffix(name,”mem”)
    local mpx = mplib.new {
      ini_version = false,
      mem_name = mem_name,
      find_file = luamplib.finder
    }
    if not mpx and type(luamplib.make) == ”function” then
      -- when i have time i’ll locate the format and dump
      mpx = luamplib.make(name,mem_name)
    end
    if mpx then
      info(”using format %s”,mem_name,false)
      return mpx, nil
    else
      return nil, { status = 99, error = ”out of memory or invalid format” }
    end
  end
```

```

else

```

These are the versions called with sufficiently recent mplib.

```
  local preamble = [[
    boolean mplib ; mplib := true ;
    let dump = endinput ;
    let normalfontsize = fontsize;
    input %s ;
  ]]

  luamplib.make = luamplib.make or function()
  end

  function luamplib.load(name,verbatim)
    local mpx = mplib.new {
      ini_version = true,
      find_file = luamplib.finder,
```

Provides `numbersystem` option since v2.4. Default value `"scaled"` can be changed by declaring `\mplibnumbersystem{double}`. See `https://github.com/lualatex/luamplib/issues/21`.

```
      math_mode = luamplib.numbersystem,
      random_seed = randomseed,
    }
```

Append our own preamble to the preamble above.

```
    local preamble = preamble .. (verbatim and "" or luamplib.mplibcodepreamble)
    if luamplib.textextlabel then
      preamble = preamble .. (verbatim and "" or luamplib.textextlabelpreamble)
    end
    local result
    if not mpx then
      result = { status = 99, error = "out of memory"}
    else
      result = mpx:execute(format(preamble, replacesuffix(name,"mp")))
    end
    luamplib.reporterror(result)
    return mpx, result
  end

end

local currentformat = "plain"

local function setformat (name) --- used in .sty
  currentformat = name
end
luamplib.setformat = setformat


luamplib.reporterror = function (result)
  if not result then
    err("no result object returned")
  else
    local t, e, l = result.term, result.error, result.log
    local log = stringgsub(t or l or "no-term","^%s+","\n")
    luamplib.lastlog = luamplib.lastlog .. "\n " .. (l or t or "no-log")
    if result.status > 0 then
      warn("%s",log)
      if result.status > 1 then
        err("%s",e or "see above messages")
      end
    end
    return log
  end
end

local function process_indeed (mpx, data, indeed)
  local converted, result = false, {}
  if mpx and data then
    result = mpx:execute(data)
    local log = luamplib.reporterror(result)
    if indeed and log then
      if luamplib.showlog then
        info("%s",luamplib.lastlog)
        luamplib.resetlastlog()
      elseif result.fig then
```

v2.6.1: now luamplib does not disregard `show` command, even when `luamplib.showlog` is false. Incidentally, it does not raise error, but just prints a warning, even if output has no figure.

```
        if stringfind(log,"\n>>") then info("%s",log) end
        converted = luamplib.convert(result)
      else
        info("%s",log)
        warn("No figure output. Maybe no beginfig/endfig")
      end
    end
  else
    err("Mem file unloadable. Maybe generated with a different version of mplib?")
  end
  return converted, result
end

```

v2.9 has introduced the concept of 'code inherit'

```
luamplib.codeinherit = false
local mplibinstances = {}
local process = function (data,indeed,verbatim)
```

workaround issue #70

```
  if not stringfind(data, begname..”beginfig%s*%([%+%-%s]*%d[%.%d%s]*%)”) then
    data = data .. ”beginfig(-1);endfig;”
  end
  local standalone, firstpass = not luamplib.codeinherit, not indeed
  local currfmt = currentformat .. (luamplib.numbersystem or ”scaled”)
  currfmt = firstpass and currfmt or (currfmt..”2”)
  local mpx = mplibinstances[currfmt]
  if standalone or not mpx then
    randomseed = firstpass and math.random(65535) or randomseed
    mpx = luamplib.load(currentformat,verbatim)
    mplibinstances[currfmt] = mpx
  end
  return process_indeed(mpx, data, indeed)
end
luamplib.process = process

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

local function convert(result, flusher)
  luamplib.flush(result, flusher)
  return true -- done
end
luamplib.convert = convert

local function pdf_startfigure(n,llx,lly,urx,ury)
```

The following line has been slightly modified by Kim.

```
  texsprint(format(”\\mplibstarttoPDF{%f}{%f}{%f}{%f}”,llx,lly,urx,ury))
end

local function pdf_stopfigure()
  texsprint(”\\mplibstoptoPDF”)
end

```

tex.tprint and catcode regime -2, as sometimes # gets doubled in the argument of pdfliteral. — modified by Kim

```
local function pdf_literalcode(fmt,...) -- table
  textprint({”\\mplibtoPDF{”},{-2,format(fmt,...)},{”}”})
end
luamplib.pdf_literalcode = pdf_literalcode

local function pdf_textfigure(font,size,text,width,height,depth)
```

The following three lines have been modified by Kim.

```
  -- if text == ”” then text = ”\0” end -- char(0) has gone
  text = text:gsub(”.”,function(c)
    return format(”\\hbox{\\char%i}”,string.byte(c)) -- kerning happens in metapost
  end)
  texsprint(format("\\mplibtextext{%s}{%f}{%s}{%s}{%f}",font,size,text,0,-( 7200/ 7227)/65536*depth))
end
luamplib.pdf_textfigure = pdf_textfigure

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)
  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
end

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

local function curved(ith,pth)
  local d = pth.left_x - ith.right_x
  if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x - d) <= bend
    d = pth.left_y - ith.right_y
    if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y - d) <= be
      return false
    end
  end
  return true
end

local function flushnormalpath(path,open)
  local pth, ith
  for i=1,#path do
    pth = path[i]
    if not ith then
      pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
    elseif curved(ith,pth) then
      pdf_literalcode("%f %f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,p
    else
      pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
    end
    ith = pth
  end
  if not open then
    local one = path[1]
    if curved(pth,one) then
      pdf_literalcode("%f %f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coord,o
    else
      pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
    end
  elseif #path == 1 then
    -- special case .. draw point
    local one = path[1]
    pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
  end
  return t
end

local function flushconcatpath(path,open)
  pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
  local pth, ith
  for i=1,#path do
    pth = path[i]
    if not ith then
      pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
    elseif curved(ith,pth) then
      local a, b = concat(ith.right_x,ith.right_y)
      local c, d = concat(pth.left_x,pth.left_y)
      pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
    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

```

Below code has been contributed by Dohyun Kim. It implements `btex` / `etex` functions.

v2.1: `textext()` is now available, which is equivalent to `TEX()` macro from TEX.mp. `TEX()` is synonym of `textext()` unless TEX.mp is loaded.

v2.2: Transparency and Shading

v2.3: `\everymplib`, `\everyendmplib`, and allows naked TeX commands.

```
local further_split_keys = {
  ["MPlibTEXboxID"] = true,
  ["sh_color_a"]    = true,
  ["sh_color_b"]    = true,
}

local function script2table(s)
  local t = {}
  for _,i in ipairs(stringexplode(s,"\13+")) do
    local k,v = stringmatch(i,"(.-)=(.*)") -- v may contain = or empty.
    if k and v and k ~= "" then
      if further_split_keys[k] then
        t[k] = stringexplode(v,":")
      else
        t[k] = v
      end
    end
  end
  return t
end

local mplibcodepreamble = [[
vardef rawtextext (expr t) =
  if unknown TEXBOX_:
    image( special "MPlibmkTEXbox="&t;
      addto currentpicture doublepath unitsquare; )
  else:
    TEXBOX_ := TEXBOX_ + 1;
    if known TEXBOX_wd_[TEXBOX_]:
      image ( addto currentpicture doublepath unitsquare
        xscaled TEXBOX_wd_[TEXBOX_]
        yscaled (TEXBOX_ht_[TEXBOX_] + TEXBOX_dp_[TEXBOX_])
        shifted (0, -TEXBOX_dp_[TEXBOX_])
        withprescript "MPlibTEXboxID=" &
          decimal TEXBOX_ & ":" &
          decimal TEXBOX_wd_[TEXBOX_] & ":" &
          decimal(TEXBOX_ht_[TEXBOX_]+TEXBOX_dp_[TEXBOX_]); )
    else:
      image( special "MPlibTEXError=1"; )
    fi
  fi
enddef;
if known context_mlib:
  defaultfont := "cmtt10";
  let infont = normalinfont;
  let fontsize = normalfontsize;
  vardef thelabel@#(expr p,z) =
    if string p :
      thelabel@#(p infont defaultfont scaled defaultscale,z)
    else :
      p shifted (z + labeloffset*mfun_laboff@# -
        (mfun_labxf@#*lrcorner p + mfun_labyf@#*ulcorner p +
        (1-mfun_labxf@#-mfun_labyf@#)*llcorner p))
    fi
  enddef;
  def graphictext primary filename =
    if (readfrom filename = EOF):
      errmessage "Please prepare '"&filename&"' in advance with"&
      " 'pstoedit -ssp -dt -f mpost yourfile.ps "&filename&"'";
    fi
    closefrom filename;
    def data_mpy_file = filename enddef;
    mfun_do_graphic_text (filename)
  enddef;
  if unknown TEXBOX_: def mfun_do_graphic_text text t = enddef; fi
else:
  vardef textext@# (text t) = rawtextext (t) enddef;
fi
def externalfigure primary filename =
  draw rawtextext("\includegraphics{"& filename &"}")
enddef;
def TEX = textext enddef;
def specialVerbatimTeX (text t) = special "MPlibVerbTeX="&t; enddef;
def normalVerbatimTeX  (text t) = special "PostMPlibVerbTeX="&t; enddef;
let VerbatimTeX = specialVerbatimTeX;
extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;" ;
extra_endfig   := extra_endfig   & " let VerbatimTeX = specialVerbatimTeX;" ;
]]
luamplib.mplibcodepreamble = mplibcodepreamble

local textextlabelpreamble = [[
primarydef s infont f = rawtextext(s) enddef;
def fontsize expr f =
  begingroup
  save size,pic; numeric size; picture pic;
  pic := rawtextext("\hskip\pdffontsize\font");
  size := xpart urcorner pic - xpart llcorner pic;
  if size = 0: 10pt else: size fi
  endgroup
enddef;
]]
luamplib.textextlabelpreamble = textextlabelpreamble

local TeX_code_t = {}
local texboxnum = { 2047 }

local function domakeTEXboxes (data)
  local num = texboxnum[1]
  texboxnum[2] = num
  local global = luamplib.globaltextext and "\\global" or ""
  if data and data.fig then
    local figures = data.fig
    for f=1, #figures do
      TeX_code_t[f] = nil
      local figure = figures[f]
      local objects = getobjects(data,figure,f)
      if objects then
        for o=1,#objects do
          local object    = objects[o]
          local prescript = object.prescript
          prescript = prescript and script2table(prescript)
          local str = prescript and prescript.MPlibmkTEXbox
          if str then
            num = num + 1
            texsprint(format("%s\\setbox%i\\hbox{%s}", global, num, str))
          end
```

`verbatimtex ... etex` before `beginfig()` is not ignored, but the TeX code inbetween is inserted before the mplib box.

```
          local texcode = prescript and prescript.MPlibVerbTeX
          if texcode and texcode ~= "" then
            TeX_code_t[f] = texcode
          end
        end
      end
    end
  end
  if luamplib.globaltextext then
    texboxnum[1] = num
  end
end

local function protect_tex_text_common (data)
  local everymplib    = texgettoks('everymplibtoks')    or ''
  local everyendmplib = texgettoks('everyendmplibtoks') or ''
  data = format("\n%s\n%s\n%s",everymplib, data, everyendmplib)
  data = data:gsub("\r","\n")

  data = data:gsub("\"[^\n]-\"", function(str)
    return str:gsub("([bem])tex"..endname,"%1"..esctex)
  end)

  data = data:gsub(btex_etex, function(str)
    return format("rawtextext(\"%s\")",protecttexcontents(str))
  end)
  data = data:gsub(verbatimtex_etex, function(str)
    return format("VerbatimTeX(\"%s\")",protecttexcontents(str))
  end)

  return data
end

```

```
local function protecttextextVerbatim(data)
  data = protect_tex_text_common(data)

  data = data:gsub("\"[^\n]-\"", function(str) -- restore string btex .. etex
    return str:gsub("([bem])"..esctex, "%1tex")
  end)

  local _,result = process(data, false)
  domakeTEXboxes(result)
  return data
end

luamplib.protecttextextVerbatim = protecttextextVerbatim

luamplib.mpxcolors = {}

local function protecttextext(data)
  data = protect_tex_text_common(data)

  data = data:gsub("\"[^\n]-\"", function(str)
    str = str:gsub("([bem])"..esctex, "%1tex")
             :gsub("%%", escpcnt)
             :gsub("{",  esclbr)
             :gsub("}",  escrbr)
             :gsub("#",  eschash)
    return format("\\detokenize{%s}",str)
  end)

  data = data:gsub("%%.-\n", "")

  local grouplevel = tex.currentgrouplevel
  luamplib.mpxcolors[grouplevel] = {}
  data = data:gsub("\\mpcolor"..endname.."(.-){(.-)}", function(opt,str)
    local cnt = #luamplib.mpxcolors[grouplevel] + 1
    luamplib.mpxcolors[grouplevel][cnt] = format(
      "\\expandafter\\mplibcolor\\csname mpxcolor%i:%i\\endcsname%s{%s}",
      grouplevel,cnt,opt,str)
    return format("\\csname mpxcolor%i:%i\\endcsname",grouplevel,cnt)
  end)

```

Next line to address bug #55

```
  data = data:gsub("([^`\\])#","%1##")

  texsprint(data)
end

luamplib.protecttextext = protecttextext

local function makeTEXboxes (data)
```

```
  data = data:gsub("##","#")
             :gsub(escpcnt,"%%")
             :gsub(esclbr,"{")
             :gsub(escrbr,"}")
             :gsub(eschash,"#")
  local _,result = process(data, false)
  domakeTEXboxes(result)
  return data
end

luamplib.makeTEXboxes = makeTEXboxes

local factor = 65536*(7227/7200)

local function processwithTEXboxes (data)
  if not data then return end
  local num = texboxnum[2]
  local prepreamble = format("TEXBOX_:=%i;\n",num)
  while true do
    num = num + 1
    local box = texgetbox(num)
    if not box then break end
    prepreamble = format(
      "%sTEXBOX_wd_[%i]:=%f;\nTEXBOX_ht_[%i]:=%f;\nTEXBOX_dp_[%i]:=%f;\n",
      prepreamble,
      num, box.width /factor,
      num, box.height/factor,
      num, box.depth /factor)
  end
  process(prepreamble .. data, true)
end
luamplib.processwithTEXboxes = processwithTEXboxes

local pdfoutput = tonumber(texget("outputmode")) or tonumber(texget("pdfoutput"))
local pdfmode = pdfoutput > 0

local function start_pdf_code()
  if pdfmode then
    pdf_literalcode("q")
  else
    texsprint("\\special{pdf:bcontent}") -- dvipdfmx
  end
end
local function stop_pdf_code()
  if pdfmode then
    pdf_literalcode("Q")
  else
    texsprint("\\special{pdf:econtent}") -- dvipdfmx
  end
end
```

```

local function putTEXboxes (object,prescript)
  local box = prescript.MPlibTEXboxID
  local n,tw,th = box[1],tonumber(box[2]),tonumber(box[3])
  if n and tw and th then
    local op = object.path
    local first, second, fourth = op[1], op[2], op[4]
    local tx, ty = first.x_coord, first.y_coord
    local sx, rx, ry, sy = 1, 0, 0, 1
    if tw ~= 0 then
      sx = (second.x_coord - tx)/tw
      rx = (second.y_coord - ty)/tw
      if sx == 0 then sx = 0.00001 end
    end
    if th ~= 0 then
      sy = (fourth.y_coord - ty)/th
      ry = (fourth.x_coord - tx)/th
      if sy == 0 then sy = 0.00001 end
    end
    start_pdf_code()
    pdf_literalcode("%f %f %f %f %f %f cm",sx,rx,ry,sy,tx,ty)
    texsprint(format("\\mplibputtextbox{%i}",n))
    stop_pdf_code()
  end
end

```

Transparency and Shading

```
local pdf_objs = {}
local token, getpageres, setpageres = newtoken or token
local pgf = { bye = "pgfutil@everybye", extgs = "pgf@sys@addpdfresource@extgs@plain" }

if pdfmode then -- repect luaotfload-colors
  getpageres = pdf.getpageresources or function() return pdf.pageresources end
  setpageres = pdf.setpageresources or function(s) pdf.pageresources = s end
else
  texsprint("\\special{pdf:obj @MPlibTr<<>>}",
            "\\special{pdf:obj @MPlibSh<<>>}")
end

-- objstr <string> => obj <number>, new <boolean>
local function update_pdfobjs (os)
  local on = pdf_objs[os]
  if on then
    return on,false
  end
  if pdfmode then
    on = pdf.immediateobj(os)
  else
    on = pdf_objs.cnt or 0
    pdf_objs.cnt = on + 1
  end
  pdf_objs[os] = on
  return on,true
end

local transparancy_modes = { [0] = ”Normal”,
  ”Normal”,       ”Multiply”,     ”Screen”,       ”Overlay”,
  ”SoftLight”,    ”HardLight”,    ”ColorDodge”,   ”ColorBurn”,
  ”Darken”,       ”Lighten”,      ”Difference”,   ”Exclusion”,
  ”Hue”,          ”Saturation”,   ”Color”,        ”Luminosity”,
  ”Compatible”,
}

local function update_tr_res(res,mode,opaq)
  local os = format(”<</BM /%s/ca %.3f/CA %.3f/AIS false>>”,mode,opaq,opaq)
  local on, new = update_pdfobjs(os)
  if new then
    if pdfmode then
      res = format(”%s/MPlibTr%i %i 0 R”,res,on,on)
    else
      if pgf.loaded then
        texsprint(format(”\\csname %s\\endcsname{/MPlibTr%i%s}”, pgf.extgs, on, os))
      else
        texsprint(format(”\\special{pdf:put @MPlibTr<</MPlibTr%i%s>>}”,on,os))
      end
    end
  end
  return res,on
end

local function tr_pdf_pageresources(mode,opaq)
  if token and pgf.bye and not pgf.loaded then
    pgf.loaded = token.create(pgf.bye).cmdname == ”assign_toks”
    pgf.bye    = pgf.loaded and pgf.bye
  end
  local res, on_on, off_on = ””, nil, nil
  res, off_on = update_tr_res(res, ”Normal”, 1)
  res, on_on  = update_tr_res(res, mode, opaq)
  if pdfmode then
    if res ~= ”” then
      if pgf.loaded then
        texsprint(format(”\\csname %s\\endcsname{%s}”, pgf.extgs, res))
      else
        local tpr, n = getpageres() or ””, 0
        tpr, n = tpr:gsub(”/ExtGState<<”, ”%1”..res)
        if n == 0 then
          tpr = format(”%s/ExtGState<<%s>>”, tpr, res)
        end
        setpageres(tpr)
      end
    end
  else
    if not pgf.loaded then
      texsprint(format("\\special{pdf:put @resources<</ExtGState @MPlibTr>>}"))
    end
  end
  return on_on, off_on
end

local shading_res

local function shading_initialize ()
  shading_res = {}
  if pdfmode and luatexbase.callbacktypes and luatexbase.callbacktypes.finish_pdffile then -- ltluatex
    local shading_obj = pdf.reserveobj()
    setpageres(format("%s/Shading %i 0 R",getpageres() or "",shading_obj))
    luatexbase.add_to_callback("finish_pdffile", function()
      pdf.immediateobj(shading_obj,format("<<%s>>",tableconcat(shading_res)))
      end, "luamplib.finish_pdffile")
    pdf_objs.finishpdf = true
  end
end

local function sh_pdfpageresources(shtype,domain,colorspace,colora,colorb,coordinates)
  if not shading_res then shading_initialize() end
  local os = format("<</FunctionType 2/Domain [ %s ]/C0 [ %s ]/C1 [ %s ]/N 1>>",
                    domain, colora, colorb)
  local funcobj = pdfmode and format("%i 0 R",update_pdfobjs(os)) or os
  os = format("<</ShadingType %i/ColorSpace /%s/Function %s/Coords [ %s ]/Extend [ true true ]/AntiAlia
              shtype, colorspace, funcobj, coordinates)
  local on, new = update_pdfobjs(os)
  if pdfmode then
    if new then
      local res = format("/MPlibSh%i %i 0 R", on, on)
      if pdf_objs.finishpdf then
        shading_res[#shading_res+1] = res
      else
        local pageres = getpageres() or ""
        if not stringfind(pageres,"/Shading<<.*>>") then
          pageres = pageres.."/Shading<<>>"
        end
        pageres = pageres:gsub("/Shading<<","%1"..res)
        setpageres(pageres)
      end
    end
  else
    if new then
      texsprint(format("\\special{pdf:put @MPlibSh<</MPlibSh%i%s>>}",on,os))
    end
    texsprint(format("\\special{pdf:put @resources<</Shading @MPlibSh>>}"))
  end
  return on
end

local function color_normalize(ca,cb)
  if #cb == 1 then
    if #ca == 4 then
      cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
    else -- #ca = 3
      cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
    end
  elseif #cb == 3 then -- #ca == 4
    cb[1], cb[2], cb[3], cb[4] = 1-cb[1], 1-cb[2], 1-cb[3], 0
  end
end

local prev_override_color

local function do_preobj_color(object,prescript)
  -- transparency
  local opaq = prescript and prescript.tr_transparency
  local tron_no, troff_no
  if opaq then
    local mode = prescript.tr_alternative or 1
    mode = transparancy_modes[tonumber(mode)]
    tron_no, troff_no = tr_pdf_pageresources(mode,opaq)
    pdf_literalcode("/MPlibTr%i gs",tron_no)
  end
  -- color
  local override = prescript and prescript.MPlibOverrideColor
  if override then
    if pdfmode then
      pdf_literalcode(override)
      override = nil
    else
      texsprint(format("\\special{color push %s}",override))
      prev_override_color = override
    end
  else
    local cs = object.color
    if cs and #cs > 0 then
      pdf_literalcode(luamplib.colorconverter(cs))
      prev_override_color = nil
    elseif not pdfmode then
      override = prev_override_color
      if override then
        texsprint(format("\\special{color push %s}",override))
      end
    end
  end
  -- shading
  local sh_type = prescript and prescript.sh_type
  if sh_type then
    local domain  = prescript.sh_domain
    local centera = stringexplode(prescript.sh_center_a)
    local centerb = stringexplode(prescript.sh_center_b)
    for _,t in pairs({centera,centerb}) do
      for i,v in ipairs(t) do
        t[i] = format("%f",v)
      end
    end
    centera = tableconcat(centera," ")
    centerb = tableconcat(centerb," ")
    local colora  = prescript.sh_color_a or {0};
    local colorb  = prescript.sh_color_b or {1};
    for _,t in pairs({colora,colorb}) do
      for i,v in ipairs(t) do
        t[i] = format("%.3f",v)
      end
    end
    if #colora > #colorb then
      color_normalize(colora,colorb)
    elseif #colorb > #colora then
      color_normalize(colorb,colora)
    end
    local colorspace
    if     #colorb == 1 then colorspace = "DeviceGray"
    elseif #colorb == 3 then colorspace = "DeviceRGB"
    elseif #colorb == 4 then colorspace = "DeviceCMYK"
    else   return troff_no,override
    end
    colora = tableconcat(colora, " ")
    colorb = tableconcat(colorb, " ")
    local shade_no
    if sh_type == "linear" then
      local coordinates = tableconcat({centera,centerb}," ")
      shade_no = sh_pdfpageresources(2,domain,colorspace,colora,colorb,coordinates)
    elseif sh_type == "circular" then
      local radiusa = format("%f",prescript.sh_radius_a)
      local radiusb = format("%f",prescript.sh_radius_b)
      local coordinates = tableconcat({centera,radiusa,centerb,radiusb}," ")
      shade_no = sh_pdfpageresources(3,domain,colorspace,colora,colorb,coordinates)
    end
    pdf_literalcode("q /Pattern cs")
    return troff_no,override,shade_no
  end
  return troff_no,override
end

```

```
local function do_postobj_color(tr,over,sh)
  if sh then
    pdf_literalcode("W n /MPlibSh%s sh Q",sh)
  end
  if over then
    texsprint("\\special{color pop}")
  end
  if tr then
    pdf_literalcode("/MPlibTr%i gs",tr)
  end
end

```

End of `btex` – `etex` and Transparency/Shading patch.

```

local function flush(result,flusher)
  if result then
    local figures = result.fig
    if figures then
      for f=1, #figures do
        info("flushing figure %s",f)
        local figure = figures[f]
        local objects = getobjects(result,figure,f)
        local fignum = tonumber(stringmatch(figure:filename(),"([%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
```

luamplib silently ignores this invalid figure for those codes that do not contain `beginfig ... endfig`. (issue #70)

```
          -- invalid
          -- pdf_startfigure(fignum,0,0,0,0)
          -- pdf_stopfigure()
        else
```

Insert `verbatimtex` code before mplib box. And prepare for those codes that will be executed afterwards.

```
          if TeX_code_t[f] then
            texsprint(TeX_code_t[f])
          end
          local TeX_code_bot = {} -- PostVerbatimTeX
          pdf_startfigure(fignum,llx,lly,urx,ury)
          start_pdf_code()
          if objects then
            local savedpath = nil
            local savedhtap = nil
            for o=1,#objects do
              local object        = objects[o]
              local objecttype    = object.type
```

Change from ConTEXt code: the following 7 lines are part of the `btex...etex` patch. Again, colors are processed at this stage. Also, we collect TEX codes that will be executed after flushing.

```
      local prescript     = object.prescript
      prescript = prescript and script2table(prescript) -- prescript is now a table
      local tr_opaq,cr_over,shade_no = do_preobj_color(object,prescript)
      if prescript and prescript.MPlibTEXboxID then
        putTEXboxes(object,prescript)
      elseif prescript and prescript.PostMPlibVerbTeX then
        TeX_code_bot[#TeX_code_bot+1] = prescript.PostMPlibVerbTeX
      elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then
        -- skip
      elseif objecttype == "start_clip" then
        local evenodd = not object.istext and object.postscript == "evenodd"
        start_pdf_code()
        flushnormalpath(object.path,t,false)
        pdf_literalcode(evenodd and "W* n" or "W n")
      elseif objecttype == "stop_clip" then
        stop_pdf_code()
        miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
      elseif objecttype == "special" then
        -- not supported
        if prescript and prescript.MPlibTEXError then
          warn("textext() anomaly. Try disabling \\mplibtextextlabel.")
        end
      elseif objecttype == "text" then
        local ot = object.transform -- 3,4,5,6,1,2
        start_pdf_code()
        pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
        pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.
        stop_pdf_code()
      else
```

Color stuffs are modified and moved to several lines above.

```
        local evenodd, collect, both = false, false, false
        local postscript = object.postscript
        if not object.istext then
          if postscript == "evenodd" then
            evenodd = true
          elseif postscript == "collect" then
            collect = true
          elseif postscript == "both" then
            both = true
          elseif postscript == "eoboth" then
            evenodd = true
            both    = true
          end
        end
        if collect then
          if not savedpath then
            savedpath = { object.path or false }
            savedhtap = { object.htap or false }
          else
            savedpath[#savedpath+1] = object.path or false
            savedhtap[#savedhtap+1] = object.htap or false
          end
        else
          local ml = object.miterlimit
          if ml and ml ~= miterlimit then
            miterlimit = ml
            pdf_literalcode("%f M",ml)
          end
          local lj = object.linejoin
          if lj and lj ~= linejoin then
            linejoin = lj
            pdf_literalcode("%i j",lj)
          end
          local lc = object.linecap
          if lc and lc ~= linecap then
            linecap = lc
            pdf_literalcode("%i J",lc)
          end
          local dl = object.dash
          if dl then
            local d = format("[%s] %i d",tableconcat(dl.dashes or {}," "),dl.offset)
            if d ~= dashed then
              dashed = d
              pdf_literalcode(dashed)
            end
          elseif dashed then
            pdf_literalcode("[] 0 d")
            dashed = false
          end
          local path = object.path
          local transformed, penwidth = false, 1
          local open = path and path[1].left_type and path[#path].right_type
          local pen = object.pen
          if pen then
            if pen.type == 'elliptical' then
              transformed, penwidth = pen_characteristics(object) -- boolean, value
              pdf_literalcode("%f w",penwidth)
              if objecttype == 'fill' then
                objecttype = 'both'
              end
            else -- calculated by mplib itself
              objecttype = 'fill'
            end
          end
          if transformed then
            start_pdf_code()
```

```
end
if path then
  if savedpath then
    for i=1,#savedpath do
      local path = savedpath[i]
      if transformed then
        flushconcatpath(path,open)
      else
        flushnormalpath(path,open)
      end
    end
    savedpath = nil
  end
  if transformed then
    flushconcatpath(path,open)
  else
    flushnormalpath(path,open)
  end
```

Change from ConTEXt code: color stuff

```
  if not shade_no then ----- conflict with shading
    if objecttype == "fill" then
      pdf_literalcode(evenodd and "h f*" or "h f")
    elseif objecttype == "outline" then
      if both then
        pdf_literalcode(evenodd and "h B*" or "h B")
      else
        pdf_literalcode(open and "S" or "h S")
      end
    elseif objecttype == "both" then
      pdf_literalcode(evenodd and "h B*" or "h B")
    end
  end
end
if transformed then
  stop_pdf_code()
end
local path = object.htap
if path then
  if transformed then
    start_pdf_code()
  end
  if savedhtap then
    for i=1,#savedhtap do
      local path = savedhtap[i]
      if transformed then
        flushconcatpath(path,open)
      else
        flushnormalpath(path,open)
      end
```

```
                      end
                      savedhtap = nil
                      evenodd   = true
                    end
                    if transformed then
                      flushconcatpath(path,open)
                    else
                      flushnormalpath(path,open)
                    end
                    if objecttype == "fill" then
                      pdf_literalcode(evenodd and "h f*" or "h f")
                    elseif objecttype == "outline" then
                      pdf_literalcode(open and "S" or "h S")
                    elseif objecttype == "both" then
                      pdf_literalcode(evenodd and "h B*" or "h B")
                    end
                    if transformed then
                      stop_pdf_code()
                    end
                  end
                end
              end
```

Added to ConTEXt code: color stuff. And execute `verbatimtex` codes.

```
              do_postobj_color(tr_opaq,cr_over,shade_no)
            end
          end
          stop_pdf_code()
          pdf_stopfigure()
          if #TeX_code_bot > 0 then
            texsprint(TeX_code_bot)
          end
        end
      end
    end
  end
end
luamplib.flush = flush

local 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
luamplib.colorconverter = colorconverter
```

2.2 TeX package

```
⟨*package⟩
```

First we need to load some packages.

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname selectfont\endcsname\relax
  \input ltluatex
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luamplib}
    [2018/01/04 v2.12.2 mplib package for LuaTeX]
  \ifx\newluafunction\@undefined
  \input ltluatex
  \fi
\fi
```

Loading of lua code.

```
\directlua{require("luamplib")}
```

Support older formats

```
\ifx\scantextokens\undefined
  \let\scantextokens\luatexscantextokens
\fi
\ifx\pdfoutput\undefined
  \let\pdfoutput\outputmode
  \protected\def\pdfliteral{\pdfextension literal}
\fi
```

Set the format for metapost.

```
\def\mplibsetformat#1{\directlua{luamplib.setformat("#1")}}
```

luamplib works in both PDF and DVI mode, but only DVIPDFMx is supported currently among a number of DVI tools. So we output a warning.

```
\ifnum\pdfoutput>0
  \let\mplibtoPDF\pdfliteral
\else
  \def\mplibtoPDF#1{\special{pdf:literal direct #1}}
  \ifcsname PackageWarning\endcsname
    \PackageWarning{luamplib}{take dvipdfmx path, no support for other dvi tools currently.}
  \else
    \write128{}
    \write128{luamplib Warning: take dvipdfmx path, no support for other dvi tools currently.}
    \write128{}
  \fi
\fi
\def\mplibsetupcatcodes{%
  %catcode'\{=12 %catcode'\}=12
  \catcode‘\#=12 \catcode‘\^=12 \catcode‘\~=12 \catcode‘\_=12
  \catcode‘\&=12 \catcode‘\$=12 \catcode‘\%=12 \catcode‘\^^M=12 \endlinechar=10
}
```

Make btex...etex box zero-metric.

```
\def\mplibputtextbox#1{\vbox to 0pt{\vss\hbox to 0pt{\raise\dp#1\copy#1\hss}}}
\newcount\mplibstartlineno
\def\mplibpostmpcatcodes{%
  \catcode‘\{=12 \catcode‘\}=12 \catcode‘\#=12 \catcode‘\%=12 }
\def\mplibreplacenewlinebr{%
  \begingroup \mplibpostmpcatcodes \mplibdoreplacenewlinebr}
\begingroup\lccode‘\~=‘\^^M \lowercase{\endgroup
  \def\mplibdoreplacenewlinebr#1^^J{\endgroup\scantextokens{{}#1~}}}
```

The Plain-specific stuff.

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname selectfont\endcsname\relax
\def\mplibreplacenewlinecs{%
  \begingroup \mplibpostmpcatcodes \mplibdoreplacenewlinecs}
\begingroup\lccode‘\~=‘\^^M \lowercase{\endgroup
  \def\mplibdoreplacenewlinecs#1^^J{\endgroup\scantextokens{\relax#1~}}}
\def\mplibcode{%
  \mplibstartlineno\inputlineno
  \begingroup
  \begingroup
  \mplibsetupcatcodes
  \mplibdocode
}
\long\def\mplibdocode#1\endmplibcode{%
  \endgroup
  \ifdefined\mplibverbatimYes
    \directlua{luamplib.tempdata\the\currentgrouplevel=luamplib.protecttextextVerbatim([===[\detokenize
    \directlua{luamplib.processwithTEXboxes(luamplib.tempdata\the\currentgrouplevel)}%
  \else
    \edef\mplibtemp{\directlua{luamplib.protecttextext([===[\unexpanded{#1}]===])}}%
    \directlua{ tex.sprint(luamplib.mpxcolors[\the\currentgrouplevel]) }%
    \directlua{luamplib.tempdata\the\currentgrouplevel=luamplib.makeTEXboxes([===[\mplibtemp]===])}%
    \directlua{luamplib.processwithTEXboxes(luamplib.tempdata\the\currentgrouplevel)}%
  \fi
  \endgroup
  \ifnum\mplibstartlineno<\inputlineno\expandafter\mplibreplacenewlinecs\fi
}
\else
```

The LaTeX-specific parts: a new environment.

```
\newenvironment{mplibcode}{%
  \global\mplibstartlineno\inputlineno
  \toks@{}\ltxdomplibcode
}{}
\def\ltxdomplibcode{%
  \begingroup
  \mplibsetupcatcodes
  \ltxdomplibcodeindeed
}
\def\mplib@mplibcode{mplibcode}
\long\def\ltxdomplibcodeindeed#1\end#2{%
  \endgroup
  \toks@\expandafter{\the\toks@#1}%
  \def\mplibtemp@a{#2}\ifx\mplib@mplibcode\mplibtemp@a
    \ifdefined\mplibverbatimYes
      \directlua{luamplib.tempdata\the\currentgrouplevel=luamplib.protecttextextVerbatim([===[\the\toks
      \directlua{luamplib.processwithTEXboxes(luamplib.tempdata\the\currentgrouplevel)}%
    \else
      \edef\mplibtemp{\directlua{luamplib.protecttextext([===[\the\toks@]===])}}%
      \directlua{ tex.sprint(luamplib.mpxcolors[\the\currentgrouplevel]) }%
      \directlua{luamplib.tempdata\the\currentgrouplevel=luamplib.makeTEXboxes([===[\mplibtemp]===])}%
      \directlua{luamplib.processwithTEXboxes(luamplib.tempdata\the\currentgrouplevel)}%
    \fi
    \end{mplibcode}%
    \ifnum\mplibstartlineno<\inputlineno
      \expandafter\expandafter\expandafter\mplibreplacenewlinebr
    \fi
  \else
    \toks@\expandafter{\the\toks@\end{#2}}\expandafter\ltxdomplibcode
  \fi
}
\fi
\def\mplibverbatim#1{%
  \begingroup
  \def\mplibtempa{#1}\def\mplibtempb{enable}%
  \expandafter\endgroup
  \ifx\mplibtempa\mplibtempb
    \let\mplibverbatimYes\relax
  \else
    \let\mplibverbatimYes\undefined
  \fi
}
```

`\everymplib` & `\everyendmplib`: macros redefining `\everymplibtoks` & `\everyendmplibtoks` respectively

```
\newtoks\everymplibtoks
\newtoks\everyendmplibtoks
\protected\def\everymplib{%
  \mplibstartlineno\inputlineno
  \begingroup
  \mplibsetupcatcodes
  \mplibdoeverymplib
}
\long\def\mplibdoeverymplib#1{%
  \endgroup
  \everymplibtoks{#1}%
  \ifnum\mplibstartlineno<\inputlineno\expandafter\mplibreplacenewlinebr\fi
}
\protected\def\everyendmplib{%
  \mplibstartlineno\inputlineno
  \begingroup
  \mplibsetupcatcodes
  \mplibdoeveryendmplib
}
\long\def\mplibdoeveryendmplib#1{%
  \endgroup
  \everyendmplibtoks{#1}%
  \ifnum\mplibstartlineno<\inputlineno\expandafter\mplibreplacenewlinebr\fi
}
\def\mpdim#1{ begingroup \the\dimexpr #1\relax\space endgroup } % gmp.sty
```

Support color/xcolor packages. User interface is: `\mpcolor{teal}` or `\mpcolor[HTML]{008080}`, for example.

```
\def\mplibcolor#1{%
  \def\set@color{\edef#1{1 withprescript "MPlibOverrideColor=\current@color"}}%
  \color
}
\def\mplibnumbersystem#1{\directlua{luamplib.numbersystem = "#1"}}
\def\mplibmakenocache#1{\mplibdomakenocache #1,*,}
\def\mplibdomakenocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdomakenocache
  \else
    \ifx*#1\else
      \directlua{luamplib.noneedtoreplace["#1.mp"]=true}%
      \expandafter\expandafter\expandafter\mplibdomakenocache
    \fi
  \fi
}
\def\mplibcancelnocache#1{\mplibdocancelnocache #1,*,}
\def\mplibdocancelnocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdocancelnocache
  \else
    \ifx*#1\else
      \directlua{luamplib.noneedtoreplace["#1.mp"]=false}%
      \expandafter\expandafter\expandafter\mplibdocancelnocache
    \fi
  \fi
}
\def\mplibcachedir#1{\directlua{luamplib.getcachedir("\unexpanded{#1}")}}
\def\mplibtextextlabel#1{%
  \begingroup
  \def\tempa{enable}\def\tempb{#1}%
  \ifx\tempa\tempb
    \directlua{luamplib.textextlabel = true}%
  \else
    \directlua{luamplib.textextlabel = false}%
  \fi
  \endgroup
}
\def\mplibcodeinherit#1{%
  \begingroup
  \def\tempa{enable}\def\tempb{#1}%
  \ifx\tempa\tempb
    \directlua{luamplib.codeinherit = true}%
  \else
    \directlua{luamplib.codeinherit = false}%
  \fi
  \endgroup
}
\def\mplibglobaltextext#1{%
  \begingroup
  \def\tempa{enable}\def\tempb{#1}%
  \ifx\tempa\tempb
    \directlua{luamplib.globaltextext = true}%
  \else
    \directlua{luamplib.globaltextext = false}%
  \fi
  \endgroup
}
```

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
}
```

input luamplib.cfg when it exists

```
\openin0=luamplib.cfg
\ifeof0 \else
  \closein0
  \input luamplib.cfg
\fi
```

That's all folks!

⟨/package⟩

3 The GNU GPL License v2

The GPL requires the complete license text to be distributed along with the code. I recommend the canonical source, instead: `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`. But if you insist on an included copy, here it is. You might want to zoom in.