

# The luamplib package

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

2024/03/29 v2.27.1

## 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 `\endmpplibcode`, and in  $\text{\LaTeX}$  in the `mpplibcode` environment.

The code is from the `luatex-mpplib.lua` and `luatex-mpplib.tex` files from ConTeXt, they have been adapted to  $\text{\LaTeX}$  and Plain by Elie Roux and Philipp Gesang, new functionalities have been added by Kim Dohyun. The changes are:

- a  $\text{\LaTeX}$  environment
- all TeX macros start by `mpplib`
- use of `luatexbase` for errors, warnings and declaration
- possibility to use `btex ... etex` to typeset TeX code. `texttext()` is a more versatile macro equivalent to `TEX()` from `TEX.mp`. `TEX()` is also allowed and is a synonym of `texttext()`.

N.B. Since v2.5, `btex ... etex` input from external `mp` files will also be processed by `luamplib`.

N.B. Since v2.20, `verbatimtex ... etex` from external `mp` files will be also processed by `luamplib`. Warning: This is a change from previous version.

Some more changes and cautions are:

**\mplibforcehmode** When this macro is declared, every mplibcode figure box will be typeset in horizontal mode, so `\centering`, `\raggedleft` etc will have effects. `\mplibnoforcehmode`, being default, reverts this setting. (Actually these commands redefine `\prependtomplibbox`. You can define this command with anything suitable before a box.)

**\mpliblegacybehavior{enable}** By default, `\mpliblegacybehavior{enable}` is already declared, in which case a `verbatimtex ... etex` that comes just before `beginfig()` is not ignored, but the  $\TeX$  code 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).

```
\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`.

By contrast,  $\TeX$  code in `VerbatimTeX(...)` or `verbatimtex ... etex` between `beginfig()` and `endfig` will be inserted after flushing out the mplib figure.

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

**\mpliblegacybehavior{disable}** If `\mpliblegacybehavior{disabled}` is declared by user, any `verbatimtex ... etex` will be executed, along with `btex ... etex`, sequentially one by one. So, some  $\TeX$  code in `verbatimtex ... etex` will have effects on `btex ... etex` codes that follows.

```
\begin{mplibcode}
beginfig(0);
draw btex ABC etex;
verbatimtex \bfseries etex;
draw btex DEF etex shifted (1cm,0); % bold face
draw btex GHI etex shifted (2cm,0); % bold face
endfig;
\end{mplibcode}
```

**\everymplib, \everyendmplib** Since v2.3, new macros `\everymplib` and `\everyendmplib` redefine the lua table containing MetaPost code which will be automatically inserted at the beginning and ending of each mplibcode.

```
\everymplib{ beginfig(0); }
```

```

\everyendmplib{ endfig; }
\mplibcode % beginfig/endfig not needed
  draw fullcircle scaled 1cm;
\endmplibcode

```

**\mpdim** Since v2.3, `\mpdim` and other raw  $\TeX$  commands are allowed inside `mplib` code. This feature is inspired by `gmp.sty` authored by Enrico Gregorio. Please refer the manual of `gmp` package for details.

```

\begin{mplibcode}
  draw origin--(.6\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.

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

From v2.26.1, `l3color` is also supported by the command `\mpcolor{color expression}`, including spot colors.

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

**\mplibtextlabel** Starting with v2.6, `\mplibtextlabel{enable}` enables string labels typeset via `texttext()` instead of `infont` operator. So, `label("my text",origin)` thereafter is exactly the same as `label(texttext("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  $\TeX$ .

**\mplibcodeinherit** 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.

**Separate instances for  $\LaTeX$  environment** v2.22 has added the support for several named MetaPost instances in  $\LaTeX$  `mplibcode` environment. Syntax is like so:

```

\begin{mplibcode}[instanceName]
  % some mp code
\end{mplibcode}

```

Behaviour is as follows.

- All the variables and functions are shared only among all the environments belonging to the same instance.
- `\mplibcodeinherit` only affects environments with no instance name set (since if a name is set, the code is intended to be reused at some point).
- From v2.27, `btex ... etex` boxes are also shared and do not require `\mplibglobaltexttext`.
- When an instance name is set, respective `\currentmpinstancename` is set.

In parallel with this functionality, v2.23 and after supports optional argument of instance name for `\everymplib` and `\everyendmplib`, affecting only those `mplibcode` environments of the same name. Unnamed `\everymplib` affects not only those instances with no name, but also those with name but with no corresponding `\everymplib`. Syntax is:

```
\everymplib[instanceName]{...}
\everyendmplib[instanceName]{...}
```

**`\mplibglobaltexttext`** Formerly, to inherit `btex ... etex` boxes as well as metapost variables, it was necessary to declare `\mplibglobaltexttext{enable}` in advance. But from v2.27, this is implicitly enabled when `\mplibcodeinherit` is true.

```
\mplibcodeinherit{enable}
%\mplibglobaltexttext{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
```

Generally speaking, it is recommended to turn `mplibglobaltexttext` always on, because it has the advantage of more efficient processing. But everything has its downside: it will waste more memory resources.

**`\mplibverbatim`** Starting with v2.11, users can issue `\mplibverbatim{enable}`, after which the contents of `mplibcode` environment will be read verbatim. As a result, except for `\mpdim` and `\mpcolor`, all other  $\TeX$  commands outside `btex ... etex` or `verbatimex ... etex` are not expanded and will be fed literally into the `mplib` process.

**`\mplibshowlog`** When `\mplibshowlog{enable}` is declared, log messages returned by `mplib` instance will be printed into the `.log` file. `\mplibshowlog{disable}` will revert this functionality. This is a  $\TeX$  side interface for `luamplib.showlog`. (v2.20.8)

**Settings regarding cache files** To support `btex ... etex` in external `.mp` files, `luamplib` inspects the content of each and every `.mp` input files and makes caches if necessary, 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 `$TEXMFVAR/luamplib_cache` or, if it's not available (mostly not writable), in the directory where output files are saved: to be specific, `$TEXMF_OUTPUT_DIRECTORY/luamplib_cache`, `./luamplib_cache`, `$TEXMFOUTPUT/luamplib_cache`, and `.` in this order. (`$TEXMF_OUTPUT_DIRECTORY` is normally the value of `--output-directory` command-line option.) This behavior 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.

**About figure box metrics** 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`.

**luamplib.cfg** At the end of package loading, `luamplib` searches `luamplib.cfg` and, if found, reads the file in automatically. Frequently used settings such as `\everymplib`, `\mplibforcehmode` or `\mplibcodeinherit` 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

```

1
2 luatexbase.provides_module {
3   name      = "luamplib",
4   version   = "2.27.1",
5   date      = "2024/03/29",
6   description = "Lua package to typeset Metapost with LuaTeX's MPLib.",
7 }
8
9 local format, abs = string.format, math.abs
10
11 local err = function(...)
```

```

12 return luatexbase.module_error ("luamplib", format(...))
13 end
14 local warn = function(...)
15 return luatexbase.module_warning("luamplib", format(...))
16 end
17 local info = function(...)
18 return luatexbase.module_info ("luamplib", format(...))
19 end
20

```

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

```

21 luamplib      = luamplib or { }
22 local luamplib = luamplib
23
24 luamplib.showlog = luamplib.showlog or false
25

```

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

```

26 local tableconcat = table.concat
27 local teksprint  = tex.sprint
28 local textprint  = tex.tprint
29
30 local texget      = tex.get
31 local texgettoks = tex.gettoks
32 local texgetbox  = tex.getbox
33 local texruntoks = tex.runtoks

```

We don’t use `tex.scantoks` anymore. See below regarding `tex.runtoks`.

```

local texscantoks = tex.scantoks

```

```

34
35 if not texruntoks then
36 err("Your LuaTeX version is too old. Please upgrade it to the latest")
37 end
38
39 local is_defined = token.is_defined
40
41 local mpplib = require ('mpplib')
42 local kpse   = require ('kpse')
43 local lfs    = require ('lfs')
44
45 local lfsattributes = lfs.attributes
46 local lfsisdir     = lfs.isdir
47 local lfsmkdir     = lfs.mkdir
48 local lfstouch     = lfs.touch
49 local iioopen      = io.open
50

```

Some helper functions, prepared for the case when `l-file` etc is not loaded.

```

51 local file = file or { }
52 local replacesuffix = file.replacesuffix or function(filename, suffix)
53 return (filename:gsub("%.[%a%d]+$", "")) .. "." .. suffix
54 end

```

```

55
56 local is_writable = file.is_writable or function(name)
57   if lfs.isdir(name) then
58     name = name .. "_luamplib_temp_file_"
59     local fh = io.open(name,"w")
60     if fh then
61       fh:close(); os.remove(name)
62       return true
63     end
64   end
65 end
66 local mk_full_path = lfs.mkdirp or lfs.mkdir or function(path)
67   local full = ""
68   for sub in path:gmatch("(/*[^\w]+)") do
69     full = full .. sub
70     lfs.mkdir(full)
71   end
72 end
73

```

btex ... etex in input .mp files will be replaced in finder. Because of the limitation of MPLib regarding make\_text, we might have to make cache files modified from input files.

```

74 local luamplibtime = kpse.find_file("luamplib.lua")
75 luamplibtime = luamplibtime and lfs.attributes(luamplibtime,"modification")
76
77 local currenttime = os.time()
78
79 local outputdir
80 if lfstouch then
81   for i,v in ipairs{'TEXMFVAR','TEXMF_OUTPUT_DIRECTORY','.', 'TEXMFOUTPUT'} do
82     local var = i == 3 and v or kpse.var_value(v)
83     if var and var ~= "" then
84       for _,vv in next, var:explode(os.type == "unix" and ":" or ";") do
85         local dir = format("%s/%s",vv,"luamplib_cache")
86         if not lfs.isdir(dir) then
87           mk_full_path(dir)
88         end
89         if is_writable(dir) then
90           outputdir = dir
91           break
92         end
93       end
94     if outputdir then break end
95   end
96 end
97 end
98 outputdir = outputdir or '.'
99
100 function luamplib.getcachedir(dir)
101   dir = dir:gsub("##","#")
102   dir = dir:gsub("^~",
103     os.type == "windows" and os.getenv("UserProfile") or os.getenv("HOME"))
104   if lfstouch and dir then

```

```

105   if lfsisdir(dir) then
106     if is_writable(dir) then
107       luamplib.cachedir = dir
108     else
109       warn("Directory '%s' is not writable!", dir)
110     end
111   else
112     warn("Directory '%s' does not exist!", dir)
113   end
114 end
115 end
116

```

Some basic MetaPost files not necessary to make cache files.

```

117 local noneedtoreplace = {
118   ["boxes.mp"] = true, -- ["format.mp"] = true,
119   ["graph.mp"] = true, ["marith.mp"] = true, ["mfplain.mp"] = true,
120   ["mpost.mp"] = true, ["plain.mp"] = true, ["rboxes.mp"] = true,
121   ["sarith.mp"] = true, ["string.mp"] = true, -- ["TEX.mp"] = true,
122   ["metafun.mp"] = true, ["metafun.mpiv"] = true, ["mp-abck.mpiv"] = true,
123   ["mp-apos.mpiv"] = true, ["mp-asnc.mpiv"] = true, ["mp-bare.mpiv"] = true,
124   ["mp-base.mpiv"] = true, ["mp-blob.mpiv"] = true, ["mp-butt.mpiv"] = true,
125   ["mp-char.mpiv"] = true, ["mp-chem.mpiv"] = true, ["mp-core.mpiv"] = true,
126   ["mp-crop.mpiv"] = true, ["mp-figs.mpiv"] = true, ["mp-form.mpiv"] = true,
127   ["mp-func.mpiv"] = true, ["mp-grap.mpiv"] = true, ["mp-grid.mpiv"] = true,
128   ["mp-grph.mpiv"] = true, ["mp-idea.mpiv"] = true, ["mp-luas.mpiv"] = true,
129   ["mp-mlib.mpiv"] = true, ["mp-node.mpiv"] = true, ["mp-page.mpiv"] = true,
130   ["mp-shap.mpiv"] = true, ["mp-step.mpiv"] = true, ["mp-text.mpiv"] = true,
131   ["mp-tool.mpiv"] = true, ["mp-cont.mpiv"] = true,
132 }
133 luamplib.noneedtoreplace = noneedtoreplace
134

```

format.mp is much complicated, so specially treated.

```

135 local function replaceformatmp(file,newfile,ofmodify)
136   local fh = ioopen(file,"r")
137   if not fh then return file end
138   local data = fh:read("*all"); fh:close()
139   fh = ioopen(newfile,"w")
140   if not fh then return file end
141   fh:write(
142     "let normalinfont = infont;\n",
143     "primarydef str infont name = rawtexttext(str) enddef;\n",
144     data,
145     "vardef Fmant_(expr x) = rawtexttext(decimal abs x) enddef;\n",
146     "vardef Fexp_(expr x) = rawtexttext(\"${\"&decimal x&\"}$\") enddef;\n",
147     "let infont = normalinfont;\n"
148   ); fh:close()
149   ifstouch(newfile,currenttime,ofmodify)
150   return newfile
151 end
152

```

Replace btex ... etex and verbatimtex ... etex in input files, if needed.

```

153 local name_b = "%f[%a_]"

```



```

154 local name_e = "%f[^%a_]"
155 local btex_etex = name_b.."btex"..name_e.."%s*(.)%s*"..name_b.."etex"..name_e
156 local verbatimetex_etex = name_b.."verbatimetex"..name_e.."%s*(.)%s*"..name_b.."etex"..name_e
157
158 local function replaceinputmpfile (name,file)
159   local ofmodify = lfsattributes(file,"modification")
160   if not ofmodify then return file end
161   local cachedir = luamplib.cachedir or outputdir
162   local newfile = name:gsub("%W","_")
163   newfile = cachedir .."/luamplib_input_"..newfile
164   if newfile and luamplibtime then
165     local nf = lfsattributes(newfile)
166     if nf and nf.mode == "file" and
167       ofmodify == nf.modification and luamplibtime < nf.access then
168       return nf.size == 0 and file or newfile
169     end
170   end
171
172   if name == "format.mp" then return replaceformatmp(file,newfile,ofmodify) end
173
174   local fh = ioopen(file,"r")
175   if not fh then return file end
176   local data = fh:read("*all"); fh:close()
177

```

“etex” must be followed by a space or semicolon as specified in Lua $\TeX$  manual, which is not the case of standalone MetaPost though.

```

178   local count,cnt = 0,0
179   data, cnt = data:gsub(btex_etex, "btex %1 etex ") -- space
180   count = count + cnt
181   data, cnt = data:gsub(verbatimetex_etex, "verbatimetex %1 etex;") -- semicolon
182   count = count + cnt
183
184   if count == 0 then
185     noneedtoreplace[name] = true
186     fh = ioopen(newfile,"w");
187     if fh then
188       fh:close()
189       lfstouch(newfile,currenttime,ofmodify)
190     end
191     return file
192   end
193
194   fh = ioopen(newfile,"w")
195   if not fh then return file end
196   fh:write(data); fh:close()
197   lfstouch(newfile,currenttime,ofmodify)
198   return newfile
199 end
200

```

As the finder function for MPLib, use the kpse library and make it behave like as if MetaPost was used. And replace it with cache files if needed. See also #74, #97.

```

201 local mpkpse
202 do

```

```

203 local exe = 0
204 while arg[exe-1] do
205     exe = exe-1
206 end
207 mpkpse = kpse.new(arg[exe], "mpost")
208 end
209
210 local special_ftype = {
211     pfb = "type1 fonts",
212     enc = "enc files",
213 }
214
215 local function finder(name, mode, ftype)
216     if mode == "w" then
217         if name and name ~= "mpout.log" then
218             kpse.record_output_file(name) -- recorder
219         end
220         return name
221     else
222         ftype = special_ftype[ftype] or ftype
223         local file = mpkpse:find_file(name, ftype)
224         if file then
225             if lfstouch and ftype == "mp" and not noneedtoreplace[name] then
226                 file = replaceinputmpfile(name, file)
227             end
228         else
229             file = mpkpse:find_file(name, name:match("%a+$"))
230         end
231         if file then
232             kpse.record_input_file(file) -- recorder
233         end
234         return file
235     end
236 end
237 luamplib.finder = finder
238

```

Create and load MPLib instances. We do not support ancient version of MPLib any more. (Don't know which version of MPLib started to support `make_text` and `run_script`; let the users find it.)

```

239 if tonumber(mplib.version()) <= 1.50 then
240     err("luamplib no longer supports mplib v1.50 or lower. "..
241         "Please upgrade to the latest version of LuaTeX")
242 end
243
244 local preamble = [[
245     boolean mplib ; mplib := true ;
246     let dump = endinput ;
247     let normalfontsize = fontsize;
248     input %s ;
249 ]]
250
251 local logatload
252 local function reporterror (result, indeed)

```

```

253 if not result then
254   err("no result object returned")
255 else
256   local t, e, l = result.term, result.error, result.log
      log has more information than term, so log first (2021/08/02)
257   local log = l or t or "no-term"
258   log = log:gsub("%(Please type a command or say 'end'%)", ""):gsub("\n+", "\n")
259   if result.status > 0 then
260     warn(log)
261     if result.status > 1 then
262       err(e or "see above messages")
263     end
264   elseif indeed then
265     local log = logatload..log

```

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.

```

266   if log:find"\n>>" then
267     warn(log)
268   elseif log:find"%g" then
269     if luamplib.showlog then
270       info(log)
271     elseif not result.fig then
272       info(log)
273     end
274   end
275   logatload = ""
276 else
277   logatload = log
278 end
279 return log
280 end
281 end
282
283 local function luamplibload (name)
284   local mpx = mplib.new {
285     ini_version = true,
286     find_file   = luamplib.finder,

```

Make use of `make_text` and `run_script`, which will co-operate with LuaTeX's `tex.runtoks`. And we provide `numbersystem` option since v2.4. Default value "scaled" can be changed by declaring `\mplibnumbersystem{double}` or `\mplibnumbersystem{decimal}`. See <https://github.com/lualatex/luamplib/issues/21>.

```

287   make_text   = luamplib.maketext,
288   run_script  = luamplib.runscript,
289   math_mode   = luamplib.numbersystem,
290   job_name    = tex.jobname,
291   random_seed = math.random(4095),
292   extensions  = 1,
293 }

```

Append our own MetaPost preamble to the preamble above.

```

294 local preamble = preamble .. luamplib.mplibcodepreamble

```

```

295 if luamplib.legacy_verbatimex then
296   preamble = preamble .. luamplib.legacyverbatimexpreable
297 end
298 if luamplib.texttextlabel then
299   preamble = preamble .. luamplib.texttextlabelpreable
300 end
301 local result
302 if not mpx then
303   result = { status = 99, error = "out of memory"}
304 else
305   result = mpx:execute(format(preamble, replacesuffix(name,"mp")))
306 end
307 reporterror(result)
308 return mpx, result
309 end
310

```

plain or metafun, though we cannot support metafun format fully.

```

311 local currentformat = "plain"
312
313 local function setformat (name)
314   currentformat = name
315 end
316 luamplib.setformat = setformat
317

```

Here, excute each mplibcode data, ie `\begin{mplibcode} ... \end{mplibcode}`.

```

318 local function process_indeed (mpx, data)
319   local converted, result = false, {}
320   if mpx and data then
321     result = mpx:execute(data)
322     local log = reporterror(result, true)
323     if log then
324       if result.fig then
325         converted = luamplib.convert(result)
326       else
327         warn("No figure output. Maybe no beginfig/endfig")
328       end
329     end
330   else
331     err("Mem file unloadable. Maybe generated with a different version of mplib?")
332   end
333   return converted, result
334 end
335

```

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

```

336 luamplib.codeinherit = false
337 local mplibinstances = {}
338 local instancename
339
340 local function process (data)

```

The workaround of issue #70 seems to be unnecessary, as we use `make_text` now.

```

if not data:find(name_b.."beginfig%s*%([%+-%s]*%d[%.%d%s]*%)" then

```

```

    data = data .. "beginfig(-1);endfig;"
end

341 local defaultinstancename = currentformat .. (luamplib.numbersystem or "scaled")
342 .. tostring(luamplib.texttextlabel) .. tostring(luamplib.legacy_verbatimtex)
343 local currfmt = instancename or defaultinstancename
344 if #currfmt == 0 then
345     currfmt = defaultinstancename
346 end
347 local mpx = mplibinstances[currfmt]
348 local standalone = false
349 if currfmt == defaultinstancename then
350     standalone = not luamplib.codeinherit
351 end
352 if mpx and standalone then
353     mpx:finish()
354 end
355 if standalone or not mpx then
356     mpx = luamplibload(currentformat)
357     mplibinstances[currfmt] = mpx
358 end
359 return process_indeed(mpx, data)
360 end
361

```

make\_text and some run\_script uses Lua $\TeX$ 's tex.runtoks, which made possible running  $\TeX$  code snippets inside \directlua.

```

362 local catlatex = luatexbase.registernumber("catcodetable@latex")
363 local catat11 = luatexbase.registernumber("catcodetable@atletter")
364

```

tex.scantoks sometimes fail to read catcode properly, especially \#, \&, or \%. After some experiment, we dropped using it. Instead, a function containing tex.script seems to work nicely.

```

    local function run_tex_code_no_use (str, cat)
        cat = cat or catlatex
        texscantoks("mplibtmtoks", cat, str)
        texruntoks("mplibtmtoks")
    end

365 local function run_tex_code (str, cat)
366     cat = cat or catlatex
367     texruntoks(function() texsprint(cat, str) end)
368 end
369

```

Prepare texttext box number containers, locals, globals and possibly instances. localid can be any number. They are local anyway. The number will be reset at the start of a new code chunk. Global boxes will use \newbox command in tex.runtoks process. This is the same when codeinherit is declared as true. Boxes of an instance will also be global, so that their tex boxes can be shared among instances of the same name.

```

370 local textboxes = {
371     locals = {}, localid = 4096,

```

```

372 globals = {}, globalid = 0,
373 }

```

For conversion of sp to bp.

```

374 local factor = 65536*(7227/7200)
375
376 local texttext_fmt = [[image(addto currentpicture doublepath unitsquare )].
377 [[xscaled %f yscaled %f shifted (0,-%f) ]].
378 [[withprescript "mplibtexboxid=%i:%f:%f"]]]
379
380 local function process_tex_text (str)
381   if str then
382     local boxtable, global
383     if instancename and instancename ~= "" then
384       texboxes[instancename] = texboxes[instancename] or {}
385       boxtable, global = texboxes[instancename], "\\global"
386     elseif luamplib.globaltexttext or luamplib.codeinherit then
387       boxtable, global = texboxes.globals, "\\global"
388     else
389       boxtable, global = texboxes.locals, ""
390     end
391     local tex_box_id = boxtable[str]
392     local box = tex_box_id and texgetbox(tex_box_id)
393     if not box then
394       if global == "" then
395         tex_box_id = texboxes.localid + 1
396         texboxes.localid = tex_box_id
397       else
398         local boxid = texboxes.globalid + 1
399         texboxes.globalid = boxid
400         run_tex_code(format(
401           [[\expandafter\newbox\csname luamplib.box.%s\endcsname]], boxid))
402         tex_box_id = tex.getcount'allocationnumber'
403       end
404       boxtable[str] = tex_box_id
405       run_tex_code(format("%s\\setbox%i\\hbox{%s}", global, tex_box_id, str))
406       box = texgetbox(tex_box_id)
407     end
408     local wd = box.width / factor
409     local ht = box.height / factor
410     local dp = box.depth / factor
411     return texttext_fmt:format(wd, ht+dp, dp, tex_box_id, wd, ht+dp)
412   end
413   return ""
414 end
415

```

Make color or xcolor's color expressions usable, with \mpcolor or mplibcolor. These commands should be used with graphical objects.

Attempt to support l3color as well.

```

416 local mplibcolorfmt = {
417   xcolor = [[\begingroup\let\XC@mcolor\relax]].
418   [[\def\set@color{\global\mplibtmptoks\expandafter{\current@color}}]].
419   [[\color%s\endgroup]],
420   l3color = [[\begingroup]].

```

```

421 [[\def\_color_select:N#1{\expandafter\_color_select:nn#1}]]..
422 [[\def\_color_backend_select:nn#1#2{\global\mplibtmp toks{#1 #2}}]]..
423 [[\def\_kernel_backend_literal:e#1{\global\mplibtmp toks\expandafter{\expanded{#1}}}}]]..
424 [[\color_select:n%s\endgroup]],
425 l3xcolor = [[\begingroup\color_if_exist:nTF%s{}}].
426 [[\def\_color_select:N#1{\expandafter\_color_select:nn#1}]]..
427 [[\def\_color_backend_select:nn#1#2{\global\mplibtmp toks{#1 #2}}]]..
428 [[\def\_kernel_backend_literal:e#1{\global\mplibtmp toks\expandafter{\expanded{#1}}}}]]..
429 [[\color_select:n%s}{\let\XC@color\relax}]]..
430 [[\def\set@color{\global\mplibtmp toks\expandafter{current@color}}]]..
431 [[\color%s}\endgroup]],
432 }
433
434 local colfmt = is_defined'color_select:n' and "l3color" or "xcolor"
435 if colfmt == "l3color" then
436   run_tex_code{
437     "\newcatcodetable\luamplibcctabexplat",
438     "\begingroup",
439     "\catcode'@=11 ",
440     "\catcode'_=11 ",
441     "\catcode':=11 ",
442     "\savecatcodetable\luamplibcctabexplat",
443     "\endgroup",
444   }
445 end
446
447 local ccexplat = luatexbase.registernumber"luamplibcctabexplat"
448
449 local function process_color (str)
450   if str then
451     if not str:find("%b{") then
452       str = format("{%s}",str)
453     end
454     local myfmt = mplibcolorfmt[colfmt]
455     if colfmt == "l3color" and (is_defined"ver@xcolor.sty" or is_defined"ver@color.sty") then
456       if str:find("%b[") then
457         myfmt = mplibcolorfmt.xcolor
458       else
459         for _,v in ipairs(str:match"{{.+}}":explode"!") do
460           if not v:find"^%s*d+%s*$") then
461             local pp = token.get_macro(format("l__color_named_%s_prop",v))
462             if not pp or pp == "" then
463               myfmt = mplibcolorfmt.xcolor
464             break
465           end
466         end
467       end
468     end
469   end
470   run_tex_code(myfmt:format(str,str,str), ccexplat or catat11)
471   local t = texgettoks"mplibtmp toks"
472   return format('1 withprescript "MplibOverrideColor=%s"', t)
473 end
474 return ""

```

```
475 end
476
```

`\mpdim` is expanded before MPLib process, so code below will not be used for `mplibcode` data. But who knows anyone would want it in `.mp` input file. If then, you can say `mplibdimen(".5\textwidth")` for example.

```
477 local function process_dimen (str)
478   if str then
479     str = str:gsub("{(.+)}", "%1")
480     run_tex_code(format([[ \mplibmptoks\expandafter{\the\dimexpr %s\relax} ]], str))
481     return format("begingroup %s endgroup", texgettoks"mp libmptoks")
482   end
483   return ""
484 end
485
```

Newly introduced method of processing `verbatimtex ... etex`. Used when `\mpliblegacybehavior{false}` is declared.

```
486 local function process_verbatimtex_text (str)
487   if str then
488     run_tex_code(str)
489   end
490   return ""
491 end
492
```

For legacy `verbatimtex` process. `verbatimtex ... etex` before `beginfig()` is not ignored, but the  $\TeX$  code is inserted just before the `mplib` box. And  $\TeX$  code inside `beginfig()` ... `endfig` is inserted after the `mplib` box.

```
493 local tex_code_pre_mplib = {}
494 luamplib.figid = 1
495 luamplib.in_the_fig = false
496
497 local function legacy_mplibcode_reset ()
498   tex_code_pre_mplib = {}
499   luamplib.figid = 1
500 end
501
502 local function process_verbatimtex_prefig (str)
503   if str then
504     tex_code_pre_mplib[luamplib.figid] = str
505   end
506   return ""
507 end
508
509 local function process_verbatimtex_infig (str)
510   if str then
511     return format('special "postmplibverbtex=%s";', str)
512   end
513   return ""
514 end
515
516 local runscript_funcs = {
517   luamplibtext    = process_tex_text,
518   luamplibcolor   = process_color,
```



```

519 luamplibdimen = process_dimen,
520 luamplibprefig = process_verbatimtex_prefig,
521 luamplibinfig = process_verbatimtex_infig,
522 luamplibverbtex = process_verbatimtex_text,
523 }
524

```

For metafun format. see issue #79.

```

525 mp = mp or {}
526 local mp = mp
527 mp.mf_path_reset = mp.mf_path_reset or function() end
528 mp.mf_finish_saving_data = mp.mf_finish_saving_data or function() end
529 mp.report = mp.report or info
530
531

```

metafun 2021-03-09 changes crashes luamplib.

```

532 catcodes = catcodes or {}
533 local catcodes = catcodes
534 catcodes.numbers = catcodes.numbers or {}
535 catcodes.numbers.ctxcatcodes = catcodes.numbers.ctxcatcodes or catlatex
536 catcodes.numbers.texcatcodes = catcodes.numbers.texcatcodes or catlatex
537 catcodes.numbers.luacatcodes = catcodes.numbers.luacatcodes or catlatex
538 catcodes.numbers.notcatcodes = catcodes.numbers.notcatcodes or catlatex
539 catcodes.numbers.vrbcatcodes = catcodes.numbers.vrbcatcodes or catlatex
540 catcodes.numbers.prtcatcodes = catcodes.numbers.prtcatcodes or catlatex
541 catcodes.numbers.txtcatcodes = catcodes.numbers.txtcatcodes or catlatex
542

```

A function from Con $\TeX$ t general.

```

543 local function mpprint(buffer,...)
544   for i=1,select("#",...) do
545     local value = select(i,...)
546     if value ~= nil then
547       local t = type(value)
548       if t == "number" then
549         buffer[#buffer+1] = format("%.16f",value)
550       elseif t == "string" then
551         buffer[#buffer+1] = value
552       elseif t == "table" then
553         buffer[#buffer+1] = "(" .. tableconcat(value,",") .. ")"
554       else -- boolean or whatever
555         buffer[#buffer+1] = tostring(value)
556       end
557     end
558   end
559 end
560
561 function luamplib.runscript (code)
562   local id, str = code:match("(.-){(.*)}")
563   if id and str then
564     local f = runscript_funcs[id]
565     if f then
566       local t = f(str)
567       if t then return t end

```

```

568 end
569 end
570 local f = loadstring(code)
571 if type(f) == "function" then
572   local buffer = {}
573   function mp.print(...)
574     mpprint(buffer,...)
575   end
576   f()
577   buffer = tableconcat(buffer)
578   if buffer and buffer ~= "" then
579     return buffer
580   end
581   buffer = {}
582   mpprint(buffer, f())
583   return tableconcat(buffer)
584 end
585 return ""
586 end
587
    make_text must be one liner, so comment sign is not allowed.
588 local function protecttexcontents (str)
589   return str:gsub("\\%", "\\0PerCent\0")
590         :gsub("%%.-\n", "")
591         :gsub("%%.-$", "")
592         :gsub("%zPerCent%z", "\\%")
593         :gsub("%s+", " ")
594 end
595
596 luamplib.legacy_verbatimex = true
597
598 function luamplib.maketext (str, what)
599   if str and str ~= "" then
600     str = protecttexcontents(str)
601     if what == 1 then
602       if not str:find("\\documentclass".name_e) and
603         not str:find("\\begin%*s*{document}") and
604         not str:find("\\documentstyle".name_e) and
605         not str:find("\\usepackage".name_e) then
606         if luamplib.legacy_verbatimex then
607           if luamplib.in_the_fig then
608             return process_verbatimex_infig(str)
609           else
610             return process_verbatimex_prefig(str)
611           end
612         else
613           return process_verbatimex_text(str)
614         end
615       end
616     else
617       return process_tex_text(str)
618     end
619   end
620   return ""

```

```

621 end
622
    Our MetaPost preambles
623 local mplibcodepreamble = [[
624 texscriptmode := 2;
625 def rawtexttext (expr t) = runscript("luamplibtext{"&t&}") enddef;
626 def mplibcolor (expr t) = runscript("luamplibcolor{"&t&}") enddef;
627 def mplibdimen (expr t) = runscript("luamplibdimen{"&t&}") enddef;
628 def VerbatimTeX (expr t) = runscript("luamplibverbtex{"&t&}") enddef;
629 if known context_mlib:
630   defaultfont := "cmtt10";
631   let infont = normalinfont;
632   let fontsize = normalfontsize;
633   vardef thelabel@#(expr p,z) =
634     if string p :
635       thelabel@#(p infont defaultfont scaled defaultscale,z)
636     else :
637       p shifted (z + labeloffset*mfun_laboff@# -
638         (mfun_labxf@#*lrcorner p + mfun_labyf@#*ulcorner p +
639         (1-mfun_labxf@#-mfun_labyf@#)*llcorner p))
640     fi
641   enddef;
642   def graphicstext primary filename =
643     if (readfrom filename = EOF):
644       errmessage "Please prepare '&filename&' in advance with"&
645         " 'pstoedit -ssp -dt -f mpost yourfile.ps '&filename&'";
646     fi
647     closefrom filename;
648     def data_mpy_file = filename enddef;
649     mfun_do_graphic_text (filename)
650   enddef;
651 else:
652   vardef texttext@# (text t) = rawtexttext (t) enddef;
653 fi
654 def externalfigure primary filename =
655   draw rawtexttext("\includegraphics{"& filename &}")
656 enddef;
657 def TEX = texttext enddef;
658 ]]
659 luamplib.mplibcodepreamble = mplibcodepreamble
660
661 local legacyverbatimmpreamble = [[
662 def specialVerbatimTeX (text t) = runscript("luamplibprefig{"&t&}") enddef;
663 def normalVerbatimTeX (text t) = runscript("luamplibinfig{"&t&}") enddef;
664 let VerbatimTeX = specialVerbatimTeX;
665 extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;"&
666   "runscript(" &ditto& "luamplib.in_the_fig=true" &ditto& ");";
667 extra_endfig := extra_endfig & " let VerbatimTeX = specialVerbatimTeX;"&
668   "runscript(" &ditto&
669   "if luamplib.in_the_fig then luamplib.figid=luamplib.figid+1 end "&
670   "luamplib.in_the_fig=false" &ditto& ");";
671 ]]
672 luamplib.legacyverbatimmpreamble = legacyverbatimmpreamble
673

```

```

674 local textxlabelpreamble = [[
675 primarydef s infont f = rawtext(s) enddef;
676 def fontsize expr f =
677   begingroup
678   save size; numeric size;
679   size := mplibdimen("1em");
680   if size = 0: 10pt else: size fi
681   endgroup
682 enddef;
683 ]]
684 luamplib.textxlabelpreamble = textxlabelpreamble
685

```

When `\mplibverbatim` is enabled, do not expand `mplibcode` data.

```

686 luamplib.verbatiminput = false
687

```

Do not expand `btex ... etex, verbatimex ... etex`, and string expressions.

```

688 local function protect_expansion (str)
689   if str then
690     str = str:gsub("\\", "!!!Control!!!")
691           :gsub("%%", "!!!Comment!!!")
692           :gsub("#", "!!!HashSign!!!")
693           :gsub("{", "!!!LBrace!!!")
694           :gsub("}", "!!!RBrace!!!")
695     return format("\\unexpanded{%s}", str)
696   end
697 end
698
699 local function unprotect_expansion (str)
700   if str then
701     return str:gsub("!!!Control!!!", "\\")
702           :gsub("!!!Comment!!!", "%")
703           :gsub("!!!HashSign!!!", "#")
704           :gsub("!!!LBrace!!!", "{")
705           :gsub("!!!RBrace!!!", "}")
706   end
707 end
708
709 luamplib.everymplib = { [""] = "" }
710 luamplib.everyendmplib = { [""] = "" }
711
712 local function process_mplibcode (data, instance)
713   instancename = instance
714   texboxes.locals, texboxes.localid = {}, 4096
715
716   This is needed for legacy behavior regarding verbatimex
717   legacy_mplibcode_reset()
718
719   local everymplib = luamplib.everymplib[instancename] or
720                     luamplib.everymplib[""]
721   local everyendmplib = luamplib.everyendmplib[instancename] or
722                         luamplib.everyendmplib[""]
723   data = format("\n%s\n%s\n", everymplib, data, everyendmplib)

```

```

723 data = data:gsub("\r","\n")
724

```

This three lines are needed for mplibverbatim mode.

```

725 if luamplib.verbatiminput then
726   data = data:gsub("\mpcolor%+{.-%b{}}", "mplibcolor(\\"%1\\""")
727   data = data:gsub("\mpdim%+{%b{}}", "mplibdimen(\\"%1\\""")
728   data = data:gsub("\mpdim%+{\\%a+}", "mplibdimen(\\"%1\\""")
729 end
730
731 data = data:gsub(btex_etex, function(str)
732   return format("btex %s etex ", -- space
733     luamplib.verbatiminput and str or protect_expansion(str))
734 end)
735 data = data:gsub(verbatimetex_etex, function(str)
736   return format("verbatimetex %s etex;", -- semicolon
737     luamplib.verbatiminput and str or protect_expansion(str))
738 end)
739

```

If not mplibverbatim, expand mplibcode data, so that users can use  $\TeX$  codes in it. It has turned out that no comment sign is allowed.

```

740 if not luamplib.verbatiminput then
741   data = data:gsub("\.-\\"", protect_expansion)
742
743   data = data:gsub("\\%", "\0PerCent\0")
744   data = data:gsub("%%.\n", "")
745   data = data:gsub("%zPerCent%z", "\\%")
746
747   run_tex_code(format("\mplibtmptoks\expandafter{\expanded{%s}}", data))
748   data = texgettoks"mplibtmptoks"

```

Next line to address issue #55

```

749 data = data:gsub("##", "#")
750 data = data:gsub("\.-\\"", unprotect_expansion)
751 data = data:gsub(btex_etex, function(str)
752   return format("btex %s etex", unprotect_expansion(str))
753 end)
754 data = data:gsub(verbatimetex_etex, function(str)
755   return format("verbatimetex %s etex", unprotect_expansion(str))
756 end)
757 end
758
759 process(data)
760 end
761 luamplib.process_mplibcode = process_mplibcode
762

```

For parsing prescript materials.

```

763 local further_split_keys = {
764   mplibtexboxid = true,
765   sh_color_a   = true,
766   sh_color_b   = true,
767 }
768
769 local function script2table(s)

```

```

770 local t = {}
771 for _,i in ipairs(s:explode("\13+")) do
772   local k,v = i:match("(.-)=(.*)") -- v may contain = or empty.
773   if k and v and k ~= "" then
774     if further_split_keys[k] then
775       t[k] = v:explode(":")
776     else
777       t[k] = v
778     end
779   end
780 end
781 return t
782 end
783

```

Codes below for inserting PDF literals are mostly from ConTeXt general, with small changes when needed.

```

784 local function getobjects(result, figure, f)
785   return figure:objects()
786 end
787
788 local function convert(result, flusher)
789   luamplib.flush(result, flusher)
790   return true -- done
791 end
792 luamplib.convert = convert
793
794 local function pdf_startfigure(n,llx,lly,urx,ury)
795   texsprint(format("\mplibstarttoPDF{%f}{%f}{%f}",llx,lly,urx,ury))
796 end
797
798 local function pdf_stopfigure()
799   texsprint("\mplibstoptoPDF")
800 end
801

```

tex.tprint with catcode regime -2, as sometimes # gets doubled in the argument of pdfliteral.

```

802 local function pdf_literalcode(fmt,...) -- table
803   textprint({"\mplibtoPDF"},{-2,format(fmt,...)},{""})
804 end
805
806 local function pdf_textfigure(font,size,text,width,height,depth)
807   text = text:gsub(".",function(c)
808     return format("\hbox{\char%i}",string.byte(c)) -- kerning happens in metapost
809   end)
810   texsprint(format("\mplibtexttext{%s}{%f}{%s}{%s}{%f}",font,size,text,0,-(7200/7227)/65536*depth))
811 end
812
813 local bend_tolerance = 131/65536
814
815 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
816
817 local function pen_characteristics(object)
818   local t = mplib.pen_info(object)

```

```

819 rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
820 divider = sx*sy - rx*ry
821 return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
822 end
823
824 local function concat(px, py) -- no tx, ty here
825   return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
826 end
827
828 local function curved(ith,pth)
829   local d = pth.left_x - ith.right_x
830   if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x - d) <= bend_tolerance t
831     d = pth.left_y - ith.right_y
832     if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y - d) <= bend_tolerance
833       return false
834     end
835   end
836   return true
837 end
838
839 local function flushnormalpath(path,open)
840   local pth, ith
841   for i=1,#path do
842     pth = path[i]
843     if not ith then
844       pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
845     elseif curved(ith,pth) then
846       pdf_literalcode("%f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,pth.y_coord)
847     else
848       pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
849     end
850     ith = pth
851   end
852   if not open then
853     local one = path[1]
854     if curved(pth,one) then
855       pdf_literalcode("%f %f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coord,one.y_coord)
856     else
857       pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
858     end
859   elseif #path == 1 then -- special case .. draw point
860     local one = path[1]
861     pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
862   end
863 end
864
865 local function flushconcatpath(path,open)
866   pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
867   local pth, ith
868   for i=1,#path do
869     pth = path[i]
870     if not ith then
871       pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
872     elseif curved(ith,pth) then

```

```

873     local a, b = concat(ith.right_x,ith.right_y)
874     local c, d = concat(pth.left_x,pth.left_y)
875     pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
876     else
877     pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
878     end
879     ith = pth
880 end
881 if not open then
882     local one = path[1]
883     if curved(pth,one) then
884         local a, b = concat(pth.right_x,pth.right_y)
885         local c, d = concat(one.left_x,one.left_y)
886         pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
887     else
888         pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
889     end
890 elseif #path == 1 then -- special case .. draw point
891     local one = path[1]
892     pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
893 end
894 end
895

```

dvipdfmx is supported, though nobody seems to use it.

```

896 local pdfoutput = tonumber(texget("outputmode")) or tonumber(texget("pdfoutput"))
897 local pdfmode = pdfoutput > 0
898
899 local function start_pdf_code()
900     if pdfmode then
901         pdf_literalcode("q")
902     else
903         texsprintf("\\special{pdf:bcontent}") -- dvipdfmx
904     end
905 end
906 local function stop_pdf_code()
907     if pdfmode then
908         pdf_literalcode("Q")
909     else
910         texsprintf("\\special{pdf:econtent}") -- dvipdfmx
911     end
912 end
913

```

Now we process hboxes created from `btex ... etex` or `textext(...)` or `TEX(...)`, all being the same internally.

```

914 local function put_tex_boxes (object,prescript)
915     local box = prescript.mplibtexboxid
916     local n,tw,th = box[1],tonumber(box[2]),tonumber(box[3])
917     if n and tw and th then
918         local op = object.path
919         local first, second, fourth = op[1], op[2], op[4]
920         local tx, ty = first.x_coord, first.y_coord
921         local sx, rx, ry, sy = 1, 0, 0, 1
922         if tw ~= 0 then

```



```

923     sx = (second.x_coord - tx)/tw
924     rx = (second.y_coord - ty)/tw
925     if sx == 0 then sx = 0.00001 end
926 end
927 if th ~= 0 then
928     sy = (fourth.y_coord - ty)/th
929     ry = (fourth.x_coord - tx)/th
930     if sy == 0 then sy = 0.00001 end
931 end
932 start_pdf_code()
933 pdf_literalcode("%f %f %f %f %f %f cm",sx,rx,ry,sy,tx,ty)
934 texsprint(format("\mplibputtextbox{i}",n))
935 stop_pdf_code()
936 end
937 end
938

```

### Colors and Transparency

```

939 local pdfmanagement = is_defined'pdfmanagement_add:nnn'
940
941 local pdf_objs = {}
942 local getpageres, setpageres
943 local pgf = { extgs = "pgf@sys@addpdfresource@extgs@plain" }
944
945 if pdfmode then
946     getpageres = pdf.getpageresources or function() return pdf.pageresources end
947     setpageres = pdf.setpageresources or function(s) pdf.pageresources = s end
948 else
949     texsprint("\special{pdf:obj @MPLibTr<<>>}",
950             "\special{pdf:obj @MPLibSh<<>>}")
951 end
952
953 local function update_pdfobjs (os)
954     local on = pdf_objs[os]
955     if on then
956         return on,false
957     end
958     if pdfmode then
959         on = pdf.immediateobj(os)
960     else
961         on = pdf_objs.cnt or 0
962         texsprint(format("\special{pdf:obj @mplibpdfobj%s %s}",on,os))
963         pdf_objs.cnt = on + 1
964     end
965     pdf_objs[os] = on
966     return on,true
967 end
968
969 local transparency_modes = { [0] = "Normal",
970     "Normal",      "Multiply",    "Screen",      "Overlay",
971     "SoftLight",   "HardLight",   "ColorDodge", "ColorBurn",
972     "Darken",      "Lighten",     "Difference",  "Exclusion",
973     "Hue",         "Saturation",  "Color",      "Luminosity",
974     "Compatible",
975 }

```

```

976
977 local function update_tr_res(res,mode,opaq)
978   local os = format("<</BM /%s/ca %.3f/CA %.3f/AIS false>>",mode,opaq,opaq)
979   local on, new = update_pdfobjs(os)
980   if new then
981     if pdfmode then
982       if pdfmanagement then
983         texsprint(ccexplat,format(
984           [[\pdfmanagement_add:nnn{Page/Resources/ExtGState}{MPLibTr%s}{%s 0 R}]], on,on))
985       else
986         local tr = format("/MPLibTr%s %s 0 R",on,on)
987         if pgf.loaded then
988           texsprint(format("\csname %s\endcsname{%s}", pgf.extgs,tr))
989         elseif is_defined"TRP@list" then
990           texsprint(catat11,{
991             [[\if@files\immediate\write\@auxout{]],
992             [[\string@g@addto@macro\string\TRP@list{]],
993             tr,
994             [[}]\fi]],
995           })
996           if not token.get_macro"TRP@list":find(tr) then
997             texsprint(catat11,[[\global\TRP@reruntrue]])
998           end
999         else
1000           res = res..tr
1001         end
1002       end
1003     else
1004       if pdfmanagement then
1005         texsprint(ccexplat,format(
1006           [[\pdfmanagement_add:nnn{Page/Resources/ExtGState}{MPLibTr%s}{@mplibpdfobj%s}]], on,on))
1007       else
1008         local tr = format("/MPLibTr%s @mplibpdfobj%s",on,on)
1009         if pgf.loaded then
1010           texsprint(format("\csname %s\endcsname{%s}", pgf.extgs,tr))
1011         else
1012           texsprint(format("\special{pdf:put @MPLibTr<<%s>>}",tr))
1013         end
1014       end
1015     end
1016   end
1017   return res,on
1018 end
1019
1020 local function tr_pdf_pageresources(mode,opaq)
1021   if pgf.loaded == nil then
1022     pgf.loaded = is_defined(pgf.extgs)
1023   end
1024   local res, on_on, off_on = "", nil, nil
1025   res, off_on = update_tr_res(res, "Normal", 1)
1026   res, on_on = update_tr_res(res, mode, opaq)
1027   if pdfmanagement or pgf.loaded or is_defined"TRP@list" then
1028     return on_on, off_on
1029   end

```

```

1030 if pdfmode then
1031   if res ~= "" then
1032     local tpr, n = getpagers() or "", 0
1033     tpr, n = tpr:gsub("/ExtGState<<", "%1"..res)
1034     if n == 0 then
1035       tpr = format("%s/ExtGState<<%s>>", tpr, res)
1036     end
1037     setpagers(tpr)
1038   end
1039 else
1040   texsprint(format("\\special{pdf:put @resources<</ExtGState @MPLibTr>>}"))
1041 end
1042 return on_on, off_on
1043 end
1044

```

Shading with metafun format. (maybe legacy way)

```

1045 local shading_res
1046
1047 local function shading_initialize ()
1048   shading_res = {}
1049   if pdfmode and luatexbase.callbacktypes.finish_pdffile then -- ltluatex
1050     local shading_obj = pdf.reserveobj()
1051     setpagers(format("%s/Shading %i 0 R", getpagers() or "", shading_obj))
1052     luatexbase.add_to_callback("finish_pdffile", function()
1053       pdf.immediateobj(shading_obj, format("<<%s>>", tableconcat(shading_res)))
1054     end, "luamplib.finish_pdffile")
1055     pdf_objs.finishpdf = true
1056   end
1057 end
1058
1059 local function sh_pdfpagersources(shtype, domain, colorspace, colora, colorb, coordinates)
1060   if not pdfmanagement and not shading_res then shading_initialize() end
1061   local os = format("<</FunctionType 2/Domain [ %s ]/C0 [ %s ]/C1 [ %s ]/N 1>>",
1062     domain, colora, colorb)
1063   local funcobj = pdfmode and format("%s 0 R", update_pdfobjs(os))
1064     or format("@mplibpdfobj%s", update_pdfobjs(os))
1065   os = format("<</ShadingType %i/ColorSpace /%s/Function %s/Coords [ %s ]/Extend [ true true ]/AntiAlias true>>",
1066     shtype, colorspace, funcobj, coordinates)
1067   local on, new = update_pdfobjs(os)
1068   if pdfmode then
1069     if new then
1070       if pdfmanagement then
1071         texsprint(ccexplat, format(
1072           [[\pdfmanagement_add:nnn{Page/Resources/Shading}{MPLibSh%s}{%s 0 R}]], on, on))
1073       else
1074         local res = format("/MPLibSh%s %s 0 R", on, on)
1075         if pdf_objs.finishpdf then
1076           shading_res[#shading_res+1] = res
1077         else
1078           local pagers = getpagers() or ""
1079           if not pagers:find("/Shading<<. *>>") then
1080             pagers = pagers.."/Shading<<>>"
1081           end
1082           pagers = pagers:gsub("/Shading<<", "%1"..res)

```

```

1083         setpagemer(pageres)
1084     end
1085 end
1086 end
1087 else
1088     if pdfmanagement then
1089         if new then
1090             texsprintf(ccexplat,format(
1091                 [[\pdfmanagement_add:nnn{Page/Resources/Shading}{MPlibSh%s}{@mplibpdfobj%s}]], on,on))
1092         end
1093     else
1094         if new then
1095             texsprintf(format("\special{pdf:put @MPlibSh<<MPlibSh%s @mplibpdfobj%s>>}",on,on))
1096         end
1097         texsprintf(format("\special{pdf:put @resources<<Shading @MPlibSh>>}"))
1098     end
1099 end
1100 return on
1101 end
1102
1103 local function color_normalize(ca,cb)
1104     if #cb == 1 then
1105         if #ca == 4 then
1106             cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
1107         else -- #ca = 3
1108             cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
1109         end
1110     elseif #cb == 3 then -- #ca == 4
1111         cb[1], cb[2], cb[3], cb[4] = 1-cb[1], 1-cb[2], 1-cb[3], 0
1112     end
1113 end
1114
1115 local prev_override_color
1116
1117 local function do_preobj_color(object,prescript)
1118     transparency
1119     local opaq = prescript and prescript.tr_transparency
1120     local tron_no, troff_no
1121     if opaq then
1122         local mode = prescript.tr_alternative or 1
1123         mode = transparency_modes[tonumber(mode)]
1124         tron_no, troff_no = tr_pdf_pageresources(mode,opaq)
1125         pdf_literalcode("/MPlibTr%i gs",tron_no)
1126     end
1127
1128     color
1129     local override = prescript and prescript.MPlibOverrideColor
1130     if override then
1131         if pdfmode then
1132             pdf_literalcode(override)
1133             override = nil
1134         else
1135             if override:find"^pdf:" then
1136                 texsprintf(format("\special{%s}",override))
1137             end
1138         end
1139     end

```

```

1134     else
1135         texsprint(format("\special{color push %s}",override))
1136     end
1137     prev_override_color = override
1138 end
1139 else
1140     local cs = object.color
1141     if cs and #cs > 0 then
1142         pdf_literalcode(luamplib.colorconverter(cs))
1143         prev_override_color = nil
1144     elseif not pdfmode then
1145         override = prev_override_color
1146         if override then
1147             texsprint(format("\special{color push %s}",override))
1148         end
1149     end
1150 end
    shading
1151 local sh_type = prescript and prescript.sh_type
1152 if sh_type then
1153     local domain = prescript.sh_domain
1154     local centera = prescript.sh_center_a:explode()
1155     local centerb = prescript.sh_center_b:explode()
1156     for _,t in pairs({centera,centerb}) do
1157         for i,v in ipairs(t) do
1158             t[i] = format("%f",v)
1159         end
1160     end
1161     centera = tableconcat(centera," ")
1162     centerb = tableconcat(centerb," ")
1163     local colora = prescript.sh_color_a or {0};
1164     local colorb = prescript.sh_color_b or {1};
1165     for _,t in pairs({colora,colorb}) do
1166         for i,v in ipairs(t) do
1167             t[i] = format("%.3f",v)
1168         end
1169     end
1170     if #colora > #colorb then
1171         color_normalize(colora,colorb)
1172     elseif #colorb > #colora then
1173         color_normalize(colorb,colora)
1174     end
1175     local colorspace
1176     if #colorb == 1 then colorspace = "DeviceGray"
1177     elseif #colorb == 3 then colorspace = "DeviceRGB"
1178     elseif #colorb == 4 then colorspace = "DeviceCMYK"
1179     else return troff_no,override
1180     end
1181     colora = tableconcat(colora," ")
1182     colorb = tableconcat(colorb," ")
1183     local shade_no
1184     if sh_type == "linear" then
1185         local coordinates = tableconcat({centera,centerb}," ")
1186         shade_no = sh_pdfpageresources(2,domain,colorspace,colora,colorb,coordinates)

```

```

1187 elseif sh_type == "circular" then
1188     local radiusa = format("%f",prescript.sh_radius_a)
1189     local radiusb = format("%f",prescript.sh_radius_b)
1190     local coordinates = tableconcat({centera,radiusa,centerb,radiusb}," ")
1191     shade_no = sh_pdfpageresources(3,domain,colorspace,colora,colorb,coordinates)
1192 end
1193 pdf_literalcode("q /Pattern cs")
1194 return troff_no,override,shade_no
1195 end
1196 return troff_no,override
1197 end
1198
1199 local function do_postobj_color(tr,over,sh)
1200 if sh then
1201 pdf_literalcode("W n /MPLibSh%s sh Q",sh)
1202 end
1203 if over then
1204 texprescript("\\special{color pop}")
1205 end
1206 if tr then
1207 pdf_literalcode("/MPLibTr%i gs",tr)
1208 end
1209 end
1210

```

Finally, flush figures by inserting PDF literals.

```

1211 local function flush(result,flusher)
1212 if result then
1213 local figures = result.fig
1214 if figures then
1215 for f=1, #figures do
1216 info("flushing figure %s",f)
1217 local figure = figures[f]
1218 local objects = getobjects(result,figure,f)
1219 local fignum = tonumber(figure:filename():match("[%d]+$") or figure:charcode() or 0)
1220 local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
1221 local bbox = figure:boundingbox()
1222 local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
1223 if urx < llx then

```

luamplib silently ignores this invalid figure for those that do not contain `beginfig ... endfig`. (issue #70) Original code of ConTeXt general was:

```

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

```

```

1224 else

```

For legacy behavior. Insert ‘pre-fig’ TeX code here, and prepare a table for ‘in-fig’ codes.

```

1225 if tex_code_pre_mplib[f] then
1226 texprescript(tex_code_pre_mplib[f])
1227 end
1228 local TeX_code_bot = {}

```

```

1229     pdf_startfigure(fignum,llx,lly,urx,ury)
1230     start_pdf_code()
1231     if objects then
1232         local savedpath = nil
1233         local savedhtap = nil
1234         for o=1,#objects do
1235             local object      = objects[o]
1236             local objecttype  = object.type

```

The following 5 lines are part of `btex...etex` patch. Again, colors are processed at this stage.

```

1237         local prescript      = object.prescript
1238         prescript = prescript and script2table(prescript) -- prescript is now a table
1239         local tr_opaq,cr_over,shade_no = do_preobj_color(object,prescript)
1240         if prescript and prescript.mplibtexboxid then
1241             put_tex_boxes(object,prescript)
1242         elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then --skip
1243         elseif objecttype == "start_clip" then
1244             local evenodd = not object.istext and object.postscript == "evenodd"
1245             start_pdf_code()
1246             flushnormalpath(object.path,false)
1247             pdf_literalcode(evenodd and "W* n" or "W n")
1248         elseif objecttype == "stop_clip" then
1249             stop_pdf_code()
1250             miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
1251         elseif objecttype == "special" then

```

Collect  $\TeX$  codes that will be executed after flushing. Legacy behavior.

```

1252         if prescript and prescript.postmplibverbtx then
1253             TeX_code_bot[#TeX_code_bot+1] = prescript.postmplibverbtx
1254         end
1255     elseif objecttype == "text" then
1256         local ot = object.transform -- 3,4,5,6,1,2
1257         start_pdf_code()
1258         pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
1259         pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
1260         stop_pdf_code()
1261     else
1262         local evenodd, collect, both = false, false, false
1263         local postscript = object.postscript
1264         if not object.istext then
1265             if postscript == "evenodd" then
1266                 evenodd = true
1267             elseif postscript == "collect" then
1268                 collect = true
1269             elseif postscript == "both" then
1270                 both = true
1271             elseif postscript == "eoboth" then
1272                 evenodd = true
1273                 both = true
1274             end
1275         end
1276         if collect then
1277             if not savedpath then
1278                 savedpath = { object.path or false }

```

```

1279         savedhtap = { object.htap or false }
1280     else
1281         savedpath[#savedpath+1] = object.path or false
1282         savedhtap[#savedhtap+1] = object.htap or false
1283     end
1284 else
1285     local ml = object.miterlimit
1286     if ml and ml ~= miterlimit then
1287         miterlimit = ml
1288         pdf_literalcode("%f M",ml)
1289     end
1290     local lj = object.linejoin
1291     if lj and lj ~= linejoin then
1292         linejoin = lj
1293         pdf_literalcode("%i j",lj)
1294     end
1295     local lc = object.linecap
1296     if lc and lc ~= linecap then
1297         linecap = lc
1298         pdf_literalcode("%i J",lc)
1299     end
1300     local dl = object.dash
1301     if dl then
1302         local d = format("[%s] %f d",tableconcat(dl.dashes or {}, " "),dl.offset)
1303         if d ~= dashed then
1304             dashed = d
1305             pdf_literalcode(dashed)
1306         end
1307     elseif dashed then
1308         pdf_literalcode("[[] 0 d")
1309         dashed = false
1310     end
1311     local path = object.path
1312     local transformed, penwidth = false, 1
1313     local open = path and path[1].left_type and path[#path].right_type
1314     local pen = object.pen
1315     if pen then
1316         if pen.type == 'elliptical' then
1317             transformed, penwidth = pen_characteristics(object) -- boolean, value
1318             pdf_literalcode("%f w",penwidth)
1319             if object.type == 'fill' then
1320                 object.type = 'both'
1321             end
1322         else -- calculated by mplib itself
1323             object.type = 'fill'
1324         end
1325     end
1326     if transformed then
1327         start_pdf_code()
1328     end
1329     if path then
1330         if savedpath then
1331             for i=1,#savedpath do
1332                 local path = savedpath[i]

```



```

1333         if transformed then
1334             flushconcatpath(path,open)
1335         else
1336             flushnormalpath(path,open)
1337         end
1338     end
1339     savedpath = nil
1340 end
1341 if transformed then
1342     flushconcatpath(path,open)
1343 else
1344     flushnormalpath(path,open)
1345 end

```

Change from ConTeXt general: there was color stuffs.

```

1346         if not shade_no then -- conflict with shading
1347             if objecttype == "fill" then
1348                 pdf_literalcode(evenodd and "h f*" or "h f")
1349             elseif objecttype == "outline" then
1350                 if both then
1351                     pdf_literalcode(evenodd and "h B*" or "h B")
1352                 else
1353                     pdf_literalcode(open and "S" or "h S")
1354                 end
1355             elseif objecttype == "both" then
1356                 pdf_literalcode(evenodd and "h B*" or "h B")
1357             end
1358         end
1359     end
1360     if transformed then
1361         stop_pdf_code()
1362     end
1363     local path = object.htap
1364     if path then
1365         if transformed then
1366             start_pdf_code()
1367         end
1368         if savedhtap then
1369             for i=1,#savedhtap do
1370                 local path = savedhtap[i]
1371                 if transformed then
1372                     flushconcatpath(path,open)
1373                 else
1374                     flushnormalpath(path,open)
1375                 end
1376             end
1377             savedhtap = nil
1378             evenodd = true
1379         end
1380         if transformed then
1381             flushconcatpath(path,open)
1382         else
1383             flushnormalpath(path,open)
1384         end
1385         if objecttype == "fill" then

```

```

1386         pdf_literalcode(evenodd and "h f*" or "h f")
1387     elseif objecttype == "outline" then
1388         pdf_literalcode(open and "S" or "h S")
1389     elseif objecttype == "both" then
1390         pdf_literalcode(evenodd and "h B*" or "h B")
1391     end
1392     if transformed then
1393         stop_pdf_code()
1394     end
1395 end
1396 end
1397 end

```

Added to ConTeXt general: color stuff. And execute legacy verbatimex code.

```

1398         do_postobj_color(tr_opaq,cr_over,shade_no)
1399     end
1400 end
1401 stop_pdf_code()
1402 pdf_stopfigure()
1403 if #TeX_code_bot > 0 then texsprint(TeX_code_bot) end
1404 end
1405 end
1406 end
1407 end
1408 end
1409 luamplib.flush = flush
1410
1411 local function colorconverter(cr)
1412     local n = #cr
1413     if n == 4 then
1414         local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
1415         return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
1416     elseif n == 3 then
1417         local r, g, b = cr[1], cr[2], cr[3]
1418         return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
1419     else
1420         local s = cr[1]
1421         return format("%.3f g %.3f G",s,s), "0 g 0 G"
1422     end
1423 end
1424 luamplib.colorconverter = colorconverter

```

## 2.2 T<sub>E</sub>X package

First we need to load some packages.

```

1425 \bgroup\expandafter\expandafter\expandafter\egroup
1426 \expandafter\ifx\cname selectfont\endcsname\relax
1427 \input ltluatex
1428 \else
1429 \NeedsTeXFormat{LaTeX2e}
1430 \ProvidesPackage{luamplib}
1431 [2024/03/29 v2.27.1 mplib package for LuaTeX]
1432 \ifx\newluafunction\@undefined
1433 \input ltluatex

```

```

1434 \fi
1435 \fi

Loading of lua code.
1436 \directlua{require("luamplib")}

Support older engine. Seems we don't need it, but no harm.
1437 \ifx\pdfoutput\undefined
1438 \let\pdfoutput\outputmode
1439 \protected\def\pdfliteral{\pdfextension literal}
1440 \fi

Unfortunately there are still packages out there that think it is a good idea to manually set \pdfoutput which defeats the above branch that defines \pdfliteral. To cover that case we need an extra check.
1441 \ifx\pdfliteral\undefined
1442 \protected\def\pdfliteral{\pdfextension literal}
1443 \fi

Set the format for metapost.
1444 \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 info.
1445 \ifnum\pdfoutput>0
1446 \let\mplibtoPDF\pdfliteral
1447 \else
1448 \def\mplibtoPDF#1{\special{pdf:literal direct #1}}
1449 \ifcsname PackageInfo\endcsname
1450 \PackageInfo{luamplib}{take dvipdfmx path, no support for other dvi tools currently.}
1451 \else
1452 \write128{}
1453 \write128{luamplib Info: take dvipdfmx path, no support for other dvi tools currently.}
1454 \write128{}
1455 \fi
1456 \fi

Make mplibcode typesetted always in horizontal mode.
1457 \def\mplibforcehmode{\let\prependtomplibbox\leavevmode}
1458 \def\mplibnoforcehmode{\let\prependtomplibbox\relax}
1459 \mplibnoforcehmode

Catcode. We want to allow comment sign in mplibcode.
1460 \def\mplibsetupcatcodes{%
1461 %catcode'\{=12 %catcode'\}=12
1462 \catcode'\#=12 \catcode'\^=12 \catcode'\~=12 \catcode'\_=12
1463 \catcode'\&=12 \catcode'\$=12 \catcode'\%=12 \catcode'\^^M=12
1464 }

Make btex...etex box zero-metric.
1465 \def\mplibputtextbox#1{\vbox to 0pt{\vss\hbox to 0pt{\raise\dp#1\copy#1\hss}}}

The Plain-specific stuff.
1466 \unless\ifcsname ver@luamplib.sty\endcsname
1467 \def\mplibcode{%
1468 \begingroup
1469 \begingroup

```

```

1470 \mplibsetupcatcodes
1471 \mplibdocode
1472 }
1473 \long\def\mplibdocode#1\endmplibcode{%
1474 \endgroup
1475 \directlua{luamplib.process_mplibcode(====[\unexpanded{#1}]===, "")}%
1476 \endgroup
1477 }
1478 \else
    The  $\LaTeX$ -specific part: a new environment.
1479 \newenvironment{mplibcode}[1][{}]{%
1480 \global\def\currentmpinstancename{#1}%
1481 \mplibmptoks{}\ltxdomplibcode
1482 }{}
1483 \def\ltxdomplibcode{%
1484 \begingroup
1485 \mplibsetupcatcodes
1486 \ltxdomplibcodeindeed
1487 }
1488 \def\mplib@mplibcode{mplibcode}
1489 \long\def\ltxdomplibcodeindeed#1\end#2{%
1490 \endgroup
1491 \mplibmptoks\expandafter{\the\mplibmptoks#1}%
1492 \def\mplibtemp@a{#2}%
1493 \ifx\mplib@mplibcode\mplibtemp@a
1494 \directlua{luamplib.process_mplibcode(====[\the\mplibmptoks]===, "\currentmpinstancename")}%
1495 \end{mplibcode}%
1496 \else
1497 \mplibmptoks\expandafter{\the\mplibmptoks\end{#2}}%
1498 \expandafter\ltxdomplibcode
1499 \fi
1500 }
1501 \fi

    User settings.
1502 \def\mplibshowlog#1{\directlua{
1503 local s = string.lower("#1")
1504 if s == "enable" or s == "true" or s == "yes" then
1505 luamplib.showlog = true
1506 else
1507 luamplib.showlog = false
1508 end
1509 }}
1510 \def\mpliblegacybehavior#1{\directlua{
1511 local s = string.lower("#1")
1512 if s == "enable" or s == "true" or s == "yes" then
1513 luamplib.legacy_verbatimex = true
1514 else
1515 luamplib.legacy_verbatimex = false
1516 end
1517 }}
1518 \def\mplibverbatim#1{\directlua{
1519 local s = string.lower("#1")
1520 if s == "enable" or s == "true" or s == "yes" then

```

```

1521     luamplib.verbatiminput = true
1522   else
1523     luamplib.verbatiminput = false
1524   end
1525 }}
1526 \newtoks\mplibmptoks
      \everymplib & \everyendmplib: macros resetting luamplib.every(end)mplib tables
1527 \protected\def\everymplib{%
1528   \begingroup
1529   \mplibsetupcatcodes
1530   \mplibdoeverymplib
1531 }
1532 \protected\def\everyendmplib{%
1533   \begingroup
1534   \mplibsetupcatcodes
1535   \mplibdoeveryendmplib
1536 }
1537 \ifcsname ver@luamplib.sty\endcsname
1538   \newcommand\mplibdoeverymplib[2][]{%
1539     \endgroup
1540     \directlua{
1541       luamplib.everymplib["#1"] = [===[\unexpanded{#2}]===]
1542     }%
1543   }
1544   \newcommand\mplibdoeveryendmplib[2][]{%
1545     \endgroup
1546     \directlua{
1547       luamplib.everyendmplib["#1"] = [===[\unexpanded{#2}]===]
1548     }%
1549   }
1550 \else
1551   \long\def\mplibdoeverymplib#1{%
1552     \endgroup
1553     \directlua{
1554       luamplib.everymplib[""] = [===[\unexpanded{#1}]===]
1555     }%
1556   }
1557   \long\def\mplibdoeveryendmplib#1{%
1558     \endgroup
1559     \directlua{
1560       luamplib.everyendmplib[""] = [===[\unexpanded{#1}]===]
1561     }%
1562   }
1563 \fi

```

Allow  $\TeX$  `dimen/color` macros. Now `runscript` does the job, so the following lines are not needed for most cases. But the macros will be expanded when they are used in another macro.

```

1564 \def\mpdim#1{ runscript("luamplibdimen{#1}") }
1565 \def\mpcolor#1#\{\domplibcolor{#1}}
1566 \def\domplibcolor#1#2{ runscript("luamplibcolor{#1{#2}}") }
      MPLib's number system. Now binary has gone away.
1567 \def\mplibnumbersystem#1{\directlua{

```

```

1568 local t = "#1"
1569 if t == "binary" then t = "decimal" end
1570 luamplib.numbersystem = t
1571 }}

Settings for .mp cache files.
1572 \def\mplibmakenocache#1{\mplibdomakenocache #1,*}
1573 \def\mplibdomakenocache#1,{%
1574 \ifx\empty#1\empty
1575 \expandafter\mplibdomakenocache
1576 \else
1577 \ifx*#1\else
1578 \directlua{luamplib.noneedtoreplace["#1.mp"]=true}%
1579 \expandafter\expandafter\expandafter\mplibdomakenocache
1580 \fi
1581 \fi
1582 }
1583 \def\mplibcancelnocache#1{\mplibdocancelnocache #1,*}
1584 \def\mplibdocancelnocache#1,{%
1585 \ifx\empty#1\empty
1586 \expandafter\mplibdocancelnocache
1587 \else
1588 \ifx*#1\else
1589 \directlua{luamplib.noneedtoreplace["#1.mp"]=false}%
1590 \expandafter\expandafter\expandafter\mplibdocancelnocache
1591 \fi
1592 \fi
1593 }
1594 \def\mplibcachedir#1{\directlua{luamplib.getcachedir("\unexpanded{#1}")}}

More user settings.
1595 \def\mplibtexttextlabel#1{\directlua{
1596 local s = string.lower("#1")
1597 if s == "enable" or s == "true" or s == "yes" then
1598 luamplib.texttextlabel = true
1599 else
1600 luamplib.texttextlabel = false
1601 end
1602 }}
1603 \def\mplibcodeinherit#1{\directlua{
1604 local s = string.lower("#1")
1605 if s == "enable" or s == "true" or s == "yes" then
1606 luamplib.codeinherit = true
1607 else
1608 luamplib.codeinherit = false
1609 end
1610 }}
1611 \def\mplibglobaltexttext#1{\directlua{
1612 local s = string.lower("#1")
1613 if s == "enable" or s == "true" or s == "yes" then
1614 luamplib.globaltexttext = true
1615 else
1616 luamplib.globaltexttext = false
1617 end
1618 }}

```

The followings are from ConTeXt general, mostly. We use a dedicated scratchbox.

```
1619 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
```

We encapsulate the literals.

```
1620 \def\mplibstarttoPDF#1#2#3#4{%
1621   \prependtomplibbox
1622   \hbox\bgroup
1623   \xdef\MPllx{#1}\xdef\MPlly{#2}%
1624   \xdef\MPurx{#3}\xdef\MPury{#4}%
1625   \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
1626   \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
1627   \parskip0pt%
1628   \leftskip0pt%
1629   \parindent0pt%
1630   \everypar{}%
1631   \setbox\mplibscratchbox\vbox\bgroup
1632   \noindent
1633 }
1634 \def\mplibstoptoPDF{%
1635   \par
1636   \egroup %
1637   \setbox\mplibscratchbox\hbox %
1638     {\hskip-\MPllx bp%
1639      \raise-\MPlly bp%
1640      \box\mplibscratchbox}%
1641   \setbox\mplibscratchbox\vbox to \MPheight
1642     {\vfill
1643      \hsize\MPwidth
1644      \wd\mplibscratchbox0pt%
1645      \ht\mplibscratchbox0pt%
1646      \dp\mplibscratchbox0pt%
1647      \box\mplibscratchbox}%
1648   \wd\mplibscratchbox\MPwidth
1649   \ht\mplibscratchbox\MPheight
1650   \box\mplibscratchbox
1651   \egroup
1652 }
```

Text items have a special handler.

```
1653 \def\mplibtexttext#1#2#3#4#5{%
1654   \begingroup
1655   \setbox\mplibscratchbox\hbox
1656     {\font\temp=#1 at #2bp%
1657      \temp
1658      #3}%
1659   \setbox\mplibscratchbox\hbox
1660     {\hskip#4 bp%
1661      \raise#5 bp%
1662      \box\mplibscratchbox}%
1663   \wd\mplibscratchbox0pt%
1664   \ht\mplibscratchbox0pt%
1665   \dp\mplibscratchbox0pt%
1666   \box\mplibscratchbox
1667   \endgroup
1668 }
```

Input luamplib.cfg when it exists.

```
1669 \openin0=luamplib.cfg
1670 \ifeof0 \else
1671 \closein0
1672 \input luamplib.cfg
1673 \fi
```

That's all folks!



# 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.

## GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things. To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

### TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

- This License applies to any program or other work which contains a notice placed by the copyright holder stating it may be distributed under the terms of this General Public License. The "Program" below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you". Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.
- You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
- You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
  - You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
  - If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be

on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

- You may copy and distribute the Program for a work based on it, under Section 1 above, provided that you also do one of the following:

- Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or
- Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or
- Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

- You may not copy, modify, sublicense, or distribute the Program except as expressly permitted under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

- Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

- If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit you to freely redistribute the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances. It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

- If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

- The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

- If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### NO WARRANTY

- BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

- IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REPAIR THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

### END OF TERMS AND CONDITIONS

## Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does.  
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail. If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) yyyy name of author  
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  
This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands show w and show c should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than show w and show c; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample, alter the names:

Yooyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989  
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.