

# The luamplib package

Hans Hagen, Taco Hoekwater, Elie Roux, Philipp Gesang and Kim Dohyun  
Maintainer: LuaLaTeX Maintainers – Support: <[lualatex-dev@tug.org](mailto:lualatex-dev@tug.org)>

2020/02/24 v2.20.5

## Abstract

Package to have metapost code typeset directly in a document with  $\text{\LaTeX}$ .

## 1 Documentation

This packages aims at providing a simple way to typeset directly metapost code in a document with  $\text{\LaTeX}$ .  $\text{\LaTeX}$  is built with the `luamplib` library, that runs metapost code. This package is basically a wrapper (in Lua) for the `Luamplib` functions and some  $\text{\TeX}$  functions to have the output of the `mp` 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  $\text{\TeX}$  `hbox` with dimensions adjusted to the metapost code.

Using this package is easy: in Plain, type your metapost code between the macros `\mp` and `\endmp`, and in  $\text{\LaTeX}$  in the `mp` environment.

The code is from the `luatest-mp`.lua and `luatest-mp`.tex files from Con $\text{\TeX}$ t, 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  $\text{\TeX}$  environment
- all  $\text{\TeX}$  macros start by `mp`
- use of `luatestbase` for errors, warnings and declaration
- possibility to use `btx ... etex` to typeset  $\text{\TeX}$  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, `btx ... 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 `\verb+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 `\verb+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{disabled}** If `\mpliblegacybehavior{disabled}` is declared by user, any `\verb+verbatimtex ... etex+` will be executed, along with `\verb+btexte ... etex+`, sequentially one by one. So, some `\TeX` code in `verbatimtex ... etex` will have effects on `btexte ... etex` codes that follows.

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

**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.

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

```
\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--(\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 `btx ... etex` as provided by gmp package. As luamplib automatically protects TeX code inbetween, `\btx` 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, `(x)spotcolor` (in PDF mode) and `xespotcolor` (in DVI mode) packages are supported as well.

**`\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>.

**Settings regarding cache files** To support `btx ... 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 `btx ... etex` command. So luamplib provides macros as follows, so that users can give instruction about files that do not require this functionality.

- `\mplibmakencache{<filename>[,<filename>, ...]}`

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

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

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

**`\mplibtexttextlabel`** Starting with v2.6, `\mplibtexttextlabel{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  $\text{\TeX}$  font. Also take care of `char` operator in the left side argument, as this might bring unpermitted characters into  $\text{\TeX}$ .

**`\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.

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

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

**`\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  $\text{\TeX}$  commands outside `btx ... etex` or `verbatimtex ... etex` are not expanded and will be fed literally into the `mplib` process.

**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` or `\mplibforcehmode` 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   = "2.20.5",
5   date      = "2020/02/24",
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(...) return luatexbase.module_error ("luamplib", format(...)) end
12 local warn = function(...) return luatexbase.module_warning("luamplib", format(...)) end
13 local info = function(...) return luatexbase.module_info  ("luamplib", format(...)) end
14

```

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

```

15 luamplib      = luamplib or { }
16 local luamplib = luamplib
17
18 luamplib.showlog = luamplib.showlog or false
19 luamplib.lastlog = ""
20

```

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

```

21 local tableconcat = table.concat
22 local tex sprint = tex.sprint
23 local texprint    = tex.tprint
24
25 local texget     = tex.get
26 local texgettoks = tex.gettoks
27 local texgetbox  = tex.getbox
28 local texruntoks = tex.runtoks

```

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

```
local texscantoks = tex.scantoks
```

```

29
30 if not texruntoks then
31   err("Your LuaTeX version is too old. Please upgrade it to the latest")
32 end
33
34 local mplib = require ('mplib')
35 local kpse  = require ('kpse')
36 local lfs   = require ('lfs')
37
38 local lfsattributes = lfs.attributes
39 local lfsisdir      = lfs.isdir
40 local lfsmkdir     = lfs.mkdir
41 local lfstouch     = lfs.touch
42 local ioopen        = io.open
43

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

44 local file = file or { }
45 local replacesuffix = file.replacesuffix or function(filename, suffix)
46   return (filename:gsub("%.[%a%d]+$","")) .. "." .. suffix
47 end
48 local stripsuffix = file.stripesuffix or function(filename)
49   return (filename:gsub("%.[%a%d]+$",""))
50 end
51
52 local is_writable = file.is_writable or function(name)
53   if lfsisdir(name) then
54     name = name .. "/_luamplib_temp_file_"
55     local fh = ioopen(name,"w")
56     if fh then
57       fh:close(); os.remove(name)
58     return true
59   end
60 end
61 end
62 local mk_full_path = lfs.mkdirs or function(path)
63   local full = ""
64   for sub in path:gmatch("(/*[^\\/]*)") do
65     full = full .. sub
66     lfsmkdir(full)
67   end
68 end
69

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.

70 local luamplibtime = kpse.find_file("luamplib.lua")
71 luamplibtime = luamplibtime and lfsattributes(luamplibtime,"modification")
72

```

```

73 local currenttime = os.time()
74
75 local outputdir
76 if lfstouch then
77   local texmfvar = kpse.expand_var('$TEXMFVAR')
78   if texmfvar and texmfvar == "" and texmfvar ~= '$TEXMFVAR' then
79     for _,dir in next, texmfvar:explode(os.type == "windows" and ";" or ":") do
80       if not lfsisdir(dir) then
81         mk_full_path(dir)
82       end
83       if is_writable(dir) then
84         local cached = format("%s/luamplib_cache",dir)
85         lfsmkdir(cached)
86         outputdir = cached
87         break
88       end
89     end
90   end
91 end
92 if not outputdir then
93   outputdir = "."
94   for _,v in ipairs(arg) do
95     local t = v:match("%-output%-directory=(.+)")
96     if t then
97       outputdir = t
98       break
99     end
100   end
101 end
102
103 function luamplib.getcachedir(dir)
104   dir = dir:gsub("##","#")
105   dir = dir:gsub("^~",
106   os.type == "windows" and os.getenv("UserProfile") or os.getenv("HOME"))
107   if lfstouch and dir then
108     if lfsisdir(dir) then
109       if is_writable(dir) then
110         luamplib.cachedir = dir
111       else
112         warn("Directory '..dir..' is not writable!")
113       end
114     else
115       warn("Directory '..dir..' does not exist!")
116     end
117   end
118 end
119

```

Some basic MetaPost files not necessary to make cache files.

```

120 local noneedtoreplace = {

```

```

121 ["boxes.mp"] = true, -- ["format.mp"] = true,
122 ["graph.mp"] = true, ["marith.mp"] = true, ["mfplain.mp"] = true,
123 ["mpost.mp"] = true, ["plain.mp"] = true, ["rboxes.mp"] = true,
124 ["sarith.mp"] = true, ["string.mp"] = true, -- ["TEX.mp"] = true,
125 ["metafun.mp"] = true, ["metafun.mppiv"] = true, ["mp-abck.mppiv"] = true,
126 ["mp-apos.mppiv"] = true, ["mp-asnc.mppiv"] = true, ["mp-bare.mppiv"] = true,
127 ["mp-base.mppiv"] = true, ["mp-blob.mppiv"] = true, ["mp-butt.mppiv"] = true,
128 ["mp-char.mppiv"] = true, ["mp-chem.mppiv"] = true, ["mp-core.mppiv"] = true,
129 ["mp-crop.mppiv"] = true, ["mp-figs.mppiv"] = true, ["mp-form.mppiv"] = true,
130 ["mp-func.mppiv"] = true, ["mp-grap.mppiv"] = true, ["mp-grid.mppiv"] = true,
131 ["mp-grph.mppiv"] = true, ["mp-idea.mppiv"] = true, ["mp-luas.mppiv"] = true,
132 ["mp-mlib.mppiv"] = true, ["mp-node.mppiv"] = true, ["mp-page.mppiv"] = true,
133 ["mp-shap.mppiv"] = true, ["mp-step.mppiv"] = true, ["mp-text.mppiv"] = true,
134 ["mp-tool.mppiv"] = true,
135 }
136 luamplib.noneedtoreplace = noneedtoreplace
137

format.mp is much complicated, so specially treated.

138 local function replaceformatmp(file,newfile,ofmodify)
139   local fh = ioopen(file,"r")
140   if not fh then return file end
141   local data = fh:read("*all"); fh:close()
142   fh = ioopen(newfile,"w")
143   if not fh then return file end
144   fh:write(
145     "let normalinfont = infont;\n",
146     "primarydef str infont name = rawtexttext(str) enddef;\n",
147     data,
148     "vardef Fmant_(expr x) = rawtexttext(decimal abs x) enddef;\n",
149     "vardef Fexp_(expr x) = rawtexttext(\"$^{\"&decimal x&\"}\"\") enddef;\n",
150     "let infont = normalinfont;\n"
151   ); fh:close()
152   lfstouch(newfile,currentTime,ofmodify)
153   return newfile
154 end
155

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

156 local name_b = "%f[%a_]"
157 local name_e = "%f[^%a_]"
158 local btex_etex = name_b.."btex"..name_e.."%"..name_b.."etex"..name_e
159 local verbatimtex_etex = name_b.."verbatimtex"..name_e.."%"..name_b.."etex"..name_e
160
161 local function replaceinputmpfile (name,file)
162   local ofmodify = lfsattributes(file,"modification")
163   if not ofmodify then return file end
164   local cachedir = luamplib.cachedir or outputdir
165   local newfile = name:gsub("%", "_")
166   newfile = cachedir .."/luamplib_input_"..newfile
167   if newfile and luamplibtime then

```

```

168     local nf = lfsattributes(newfile)
169     if nf and nf.mode == "file" and
170         ofmodify == nf.modification and luamplibtime < nf.access then
171         return nf.size == 0 and file or newfile
172     end
173 end
174
175 if name == "format.mp" then return replaceformatmp(file,newfile,ofmodify) end
176
177 local fh = ioopen(file,"r")
178 if not fh then return file end
179 local data = fh:read("*all"); fh:close()
180

```

"etex" must be followed by a space or semicolon as specified in *LuaTeX* manual, which is not the case of standalone MetaPost though.

```

181 local count,cnt = 0,0
182 data, cnt = data:gsub(btex_etex, "btex %1 etex ") -- space
183 count = count + cnt
184 data, cnt = data:gsub(verbatimtex_etex, "verbatimtex %1 etex;") -- semicolon
185 count = count + cnt
186
187 if count == 0 then
188     noneedtoreplace[name] = true
189     fh = ioopen(newfile,"w");
190     if fh then
191         fh:close()
192         lfstouch(newfile,currentTime,ofmodify)
193     end
194     return file
195 end
196
197 fh = ioopen(newfile,"w")
198 if not fh then return file end
199 fh:write(data); fh:close()
200 lfstouch(newfile,currentTime,ofmodify)
201 return newfile
202 end
203

```

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.

```

204 local mpkpse = kpse.new(arg[0], "mpost")
205
206 local special_ftype = {
207     pfb = "type1 fonts",
208     enc = "enc files",
209 }
210
211 local function finder(name, mode, ftype)

```

```

212   if mode == "w" then
213     return name
214   else
215     ftype = special_ftype[ftype] or ftype
216     local file = mpkse:find_file(name,ftype)
217     if file then
218       if not lfstouch or ftype ~= "mp" or noneedtoreplace[name] then
219         return file
220       end
221       return replaceinputmpfile(name,file)
222     end
223     return mpkse:find_file(name, name:match("%a+$"))
224   end
225 end
226 luamplib.finder = finder
227

```

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.)

```

228 if tonumber(mlib.version()) <= 1.50 then
229   err("luamplib no longer supports mplib v1.50 or lower. ...
230   "Please upgrade to the latest version of LuaTeX")
231 end
232
233 local preamble = [[
234   boolean mplib ; mplib := true ;
235   let dump = endinput ;
236   let normalfontsize = fontsize;
237   input %s ;
238 ]]
239
240 local function luamplibresetlastlog()
241   luamplib.lastlog = ""
242 end
243
244 local function reporterror (result)
245   if not result then
246     err("no result object returned")
247   else
248     local t, e, l = result.term, result.error, result.log
249     local log = t or l or "no-term"
250     log = log:gsub("^%s+", "\n")
251     luamplib.lastlog = luamplib.lastlog .. "\n" .. (l or t or "no-log")
252     if result.status > 0 then
253       warn("%s",log)
254       if result.status > 1 then
255         err("%s",e or "see above messages")
256       end
257     end

```

```

258     return log
259   end
260 end
261
262 local function luamplibload (name)
263   local mpx = mpplib.new {
264     ini_version = true,
265     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 `\mpplibnumbersystem{double}` or `\mpplibnumbersystem{decimal}`. See <https://github.com/lualatex/luamplib/issues/21>.

```

266   make_text    = luamplib.maketext,
267   run_script  = luamplib.runscript,
268   math_mode   = luamplib.numbersystem,
269   extensions  = 1,
270 }

```

Append our own MetaPost preamble to the preamble above.

```

271 local preamble = preamble .. luamplib.mpplibcodepreamble
272 if luamplib.legacy_verbatimtex then
273   preamble = preamble .. luamplib.legacyverbatimtexpreamble
274 end
275 if luamplib.textextlabel then
276   preamble = preamble .. luamplib.textextlabelpreamble
277 end
278 local result
279 if not mpx then
280   result = { status = 99, error = "out of memory" }
281 else
282   result = mpx:execute(format(preamble, replacesuffix(name,"mp")))
283 end
284 reporterror(result)
285 return mpx, result
286 end
287

```

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

```

288 local currentformat = "plain"
289
290 local function setformat (name)
291   currentformat = name
292 end
293 luamplib.setformat = setformat
294

```

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

```

295 local function process_indeed (mpx, data)
296   local converted, result = false, {}
297   if mpx and data then

```

```

298     result = mpx:execute(data)
299     local log = reporterror(result)
300     if log then
301       if luamplib.showlog then
302         info("%s",luamplib.lastlog)
303         luamplibresetlastlog()
304       elseif result.fig then
305         if log:find("\n>>") then info("%s",log) end
306         converted = luamplib.convert(result)
307       else
308         info("%s",log)
309         warn("No figure output. Maybe no beginfig/endfig")
310       end
311     end
312   else
313     err("Mem file unloadable. Maybe generated with a different version of mpilib?")
314   end
315   return converted, result
316 end
317

v2.9 has introduced the concept of "code inherit"
318 luamplib.codeinherit = false
319 local mpilibinstances = {}
320
321 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

322 local standalone = not luamplib.codeinherit
323 local currfmt = currentformat .. (luamplib.numberformat or "scaled")
324   .. tostring(luamplib.texlabel) .. tostring(luamplib.legacy_verbatimtex)
325 local mpx = mpilibinstances[currfmt]
326 if mpx and standalone then
327   mpx:finish()
328 end
329 if standalone or not mpx then
330   mpx = luamplibload(currentformat)
331   mpilibinstances[currfmt] = mpx
332 end
333 return process_indeed(mpx, data)
334 end
335

```

`make_text` and some `run_script` uses LuaTeX's `tex.runtoks`, which made possible running TeX code snippets inside `\directlua`.

```
336 local catlatex = luatexbase.registernumber("catcodetable@latex")
337 local catat11  = luatexbase.registernumber("catcodetable@atletter")
338
```

`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("mplibtmptoks", cat, str)
    texruntoks("mplibtmptoks")
end

339 local function run_tex_code (str, cat)
340     cat = cat or catlatex
341     texruntoks(function() texprint(cat, str) end)
342 end
343
```

Indefinite number of boxes are needed for `btx ... etex`. So starts at somewhat huge number of box registry. Of course, this may conflict with other packages using many many boxes. (When `codeinheret` feature is enabled, boxes must be globally defined.) But I don't know any reliable way to escape this danger.

```
344 local tex_box_id = 2047
```

For conversion of `sp` to `bp`.

```
345 local factor = 65536*(7227/7200)
346
347 local texttext_fmt = [[image(addto currentpicture doublepath unitsquare )]..
348   [[xscaled %f yscaled %f shifted (0,-%f) ]][..]
349   [[withprescript "mplibtexboxid=%i:%f:%f"]]]
350
351 local function process_tex_text (str)
352     if str then
353         tex_box_id = tex_box_id + 1
354         local global = luamplib.globaltexttext and "\global" or ""
355         run_tex_code(format("%s\\setbox%i\\hbox{%s}", global, tex_box_id, str))
356         local box = texgetbox(tex_box_id)
357         local wd = box.width / factor
358         local ht = box.height / factor
359         local dp = box.depth / factor
360         return texttext_fmt:format(wd, ht+dp, dp, tex_box_id, wd, ht+dp)
361     end
362     return ""
363 end
364
```

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

```

365 local mplibcolor_fmt = [[\begingroup\let\XC@mcolor\relax]..
366   [[\def\set@color{\global\mplibmptoks\expandafter{\current@color}}]]..
367   [[\color %s \endgroup]]
368
369 local function process_color (str)
370   if str then
371     if not str:find("{}") then
372       str = format("%s",str)
373     end
374     run_tex_code(mplibcolor_fmt:format(str), catat11)
375     return format('1 withprescript "MPlibOverrideColor=%s"', texgettoks"mplibmptoks")
376   end
377   return ""
378 end
379

```

`\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.

```

380 local function process_dimen (str)
381   if str then
382     str = str:gsub("((.))","%1")
383     run_tex_code(format([[\mplibmptoks\expandafter{\the\dimexpr %s\relax}]], str))
384     return format("begingroup %s endgroup", texgettoks"mplibmptoks")
385   end
386   return ""
387 end
388

```

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

```

389 local function process_verbatimtex_text (str)
390   if str then
391     run_tex_code(str)
392   end
393   return ""
394 end
395

```

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.

```

396 local tex_code_pre_mplib = {}
397 luamplib.figid = 1
398 luamplib.in_the_fig = false
399
400 local function legacy_mplibcode_reset ()
401   tex_code_pre_mplib = {}

```

```

402 luamplib.figid = 1
403 end
404
405 local function process_verbatimtex_prefig (str)
406   if str then
407     tex_code_pre_mplib[luamplib.figid] = str
408   end
409   return ""
410 end
411
412 local function process_verbatimtex_infig (str)
413   if str then
414     return format('special "postmplibverbtex=%s";', str)
415   end
416   return ""
417 end
418
419 local runscript_funcs = {
420   luamplibtext    = process_tex_text,
421   luamplibcolor   = process_color,
422   luamplibdimen   = process_dimen,
423   luamplibprefig  = process_verbatimtex_prefig,
424   luamplibinfig   = process_verbatimtex_infig,
425   luamplibverbtex = process_verbatimtex_text,
426 }
427

For metafun format. see issue #79.

428 mp = mp or {}
429 local mp = mp
430 mp.mf_path_reset = mp.mf_path_reset or function() end
431 mp.mf_finish_saving_data = mp.mf_finish_saving_data or function() end
432

A function from ConTeXt general.

433 local function mpprint(buffer,...)
434   for i=1,select("#",...) do
435     local value = select(i,...)
436     if value ~= nil then
437       local t = type(value)
438       if t == "number" then
439         buffer[#buffer+1] = format("%.16f",value)
440       elseif t == "string" then
441         buffer[#buffer+1] = value
442       elseif t == "table" then
443         buffer[#buffer+1] = "(" .. tableconcat(value,",") .. ")"
444       else -- boolean or whatever
445         buffer[#buffer+1] = tostring(value)
446       end
447     end
448   end

```

```

449 end
450
451 function luamplib.runscript (code)
452   local id, str = code:match("(.-){(.+)}")
453   if id and str and str ~= "" then
454     local f = runscript_funcs[id]
455     if f then
456       local t = f(str)
457       if t then return t end
458     end
459   end
460   local f = loadstring(code)
461   if type(f) == "function" then
462     local buffer = {}
463     function mp.print(...)
464       mpprint(buffer,...)
465     end
466     f()
467     return tableconcat(buffer,"")
468   end
469   return ""
470 end
471

make_text must be one liner, so comment sign is not allowed.

472 local function protecttexcontents (str)
473   return str:gsub("\\%%", "\0PerCent\0")
474           :gsub("%%.-\n", "")
475           :gsub("%%.-$", "")
476           :gsub("%zPerCent%z", "\\\%")
477           :gsub("%s+", " ")
478 end
479
480 luamplib.legacy_verbatimtex = true
481
482 function luamplib.maketext (str, what)
483   if str and str ~= "" then
484     str = protecttexcontents(str)
485     if what == 1 then
486       if not str:find("\\documentclass"..name_e) and
487           not str:find("\\begin%s*{document}") and
488           not str:find("\\documentstyle"..name_e) and
489           not str:find("\\usepackage"..name_e) then
490         if luamplib.legacy_verbatimtex then
491           if luamplib.in_the_fig then
492             return process_verbatimtex_infig(str)
493           else
494             return process_verbatimtex_prefig(str)
495           end
496         else

```

```

497         return process_verbatimtex_text(str)
498     end
499   end
500 else
501   return process_tex_text(str)
502 end
503 end
504 return ""
505 end
506

```

Our MetaPost preambles

```

507 local mplibcodepreamble = [[
508 texscriptmode := 2;
509 def rawtexttext (expr t) = runscript("luamplibtext{&t&}") enddef;
510 def mplibcolor (expr t) = runscript("luamplibcolor{&t&}") enddef;
511 def mplibdimen (expr t) = runscript("luamplibdimen{&t&}") enddef;
512 def VerbatimTeX (expr t) = runscript("luamplibverbtex{&t&}") enddef;
513 if known context_mlib:
514   defaultfont := "cmtt10";
515   let infont = normalinfon;
516   let fontsize = normalfontsize;
517   vardef thelabel@#(expr p,z) =
518     if string p :
519       thelabel@#(p infont defaultfont scaled defaultscale,z)
520     else :
521       p shifted (z + labeloffset*mfun_laboff@# -
522                   (mfun_labxf@#*lrcorner p + mfun_labyf@#*ulcorner p +
523                    (1-mfun_labxf@#-mfun_labyf@#)*llcorner p))
524     fi
525   enddef;
526   def graphictext primary filename =
527     if (readfrom filename = EOF):
528       errmessage "Please prepare '&filename&' in advance with"-
529                  "'pstodeit -ssp -dt -f mpost yourfile.ps &filename&'";
530     fi
531     closefrom filename;
532     def data_mpy_file = filename enddef;
533     mfun_do_graphic_text (filename)
534   enddef;
535 else:
536   vardef texttext@# (text t) = rawtexttext (t) enddef;
537 fi
538 def externalfigure primary filename =
539   draw rawtexttext("\includegraphics{& filename &}")
540 enddef;
541 def TEX = texttext enddef;
542 ]]
543 luamplib.mplibcodepreamble = mplibcodepreamble
544

```

```

545 local legacyverbatimtexpreamble = []
546 def specialVerbatimTeX (text t) = runscript("luamplibprefig{&t&}"") enddef;
547 def normalVerbatimTeX (text t) = runscript("luamplibinfig{&t&}"") enddef;
548 let VerbatimTeX = specialVerbatimTeX;
549 extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;"&
550 "runscript(" &ditto& "luamplib.in_the_fig=true" &ditto& ");";
551 extra_endfig := extra_endfig & " let VerbatimTeX = specialVerbatimTeX;"&
552 "runscript(" &ditto&
553 "luamplib.in_the_fig=false luamplib.figid=luamplib.figid+1" &ditto& ");";
554 ]]
555 luamplib.legacyverbatimtexpreamble = legacyverbatimtexpreamble
556
557 local texttextlabelpreamble = []
558 primarydef s infont f = rawtexttext(s) enddef;
559 def fontsize expr f =
560 begingroup
561 save size; numeric size;
562 size := mpilibdimen("1em");
563 if size = 0: 10pt else: size fi
564 endgroup
565 enddef;
566 ]]
567 luamplib.texttextlabelpreamble = texttextlabelpreamble
568

When \mplibverbatim is enabled, do not expand \mplibcode data.
569 luamplib.verbatiminput = false
570

Do not expand \btx ... \etx, \verbatimtex ... \etex, and string expressions.
571 local function protect_expansion (str)
572 if str then
573 str = str:gsub("\\", "!!!Control!!!")
574 :gsub("%", "!!!Comment!!!")
575 :gsub("#", "!!!HashSign!!!")
576 :gsub("{", "!!!LBrace!!!")
577 :gsub("}", "!!!RBrace!!!")
578 return format("\unexpanded{\%s}", str)
579 end
580 end
581
582 local function unprotect_expansion (str)
583 if str then
584 return str:gsub("!!!Control!!!", "\\")
585 :gsub("!!!Comment!!!", "%")
586 :gsub("!!!HashSign!!!", "#")
587 :gsub("!!!LBrace!!!", "{")
588 :gsub("!!!RBrace!!!", "}")
589 end
590 end
591

```

```

592 local function process_mplibcode (data)
      This is needed for legacy behavior regarding verbatimtex
593   legacy_mplibcode_reset()
594
595   local everymplib    = texgettoks'everymplibtoks'  or ''
596   local everyendmplib = texgettoks'everyendmplibtoks' or ''
597   data = format("\n%s\n%s\n%s\n", everymplib, data, everyendmplib)
598   data = data:gsub("\r","\n")
599
600   data = data:gsub("\mpcolor%s+(-%b{})","mplibcolor(\"%1\")")
601   data = data:gsub("\mpdim%s+(%b{})", "mplibdimen(\"%1\")")
602   data = data:gsub("\mpdim%s+(\%a+)","mplibdimen(\"%1\")")
603
604   data = data:gsub(btex_etex, function(str)
605     return format("btex %s etex ", -- space
606                   luamplib.verbatiminput and str or protect_expansion(str))
607   end)
608   data = data:gsub(verbatimtex_etex, function(str)
609     return format("verbatimtex %s etex; ", -- semicolon
610                   luamplib.verbatiminput and str or protect_expansion(str)))
611   end)
612

```

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.

```

613 if not luamplib.verbatiminput then
614   data = data:gsub("\%. -\"", protect_expansion)
615
616   data = data:gsub("\%%", "\0PerCent\0")
617   data = data:gsub("%.-\n", "")
618   data = data:gsub("%zPerCent%z", "\%%")
619
620   run_tex_code(format("\mplibmptoks\expanded{\%s}",data))
621   data = texgettoks"mplibmptoks"

```

Next line to address issue #55

```

622   data = data:gsub("##", "#")
623   data = data:gsub("\%. -\"", unprotect_expansion)
624   data = data:gsub(btex_etex, function(str)
625     return format("btex %s etex", unprotect_expansion(str))
626   end)
627   data = data:gsub(verbatimtex_etex, function(str)
628     return format("verbatimtex %s etex", unprotect_expansion(str)))
629   end)
630 end
631
632 process(data)
633 end
634 luamplib.process_mplibcode = process_mplibcode
635

```

For parsing prescript materials.

```
636 local further_split_keys = {  
637   mplibtexboxid = true,  
638   sh_color_a    = true,  
639   sh_color_b    = true,  
640 }  
641  
642 local function script2table(s)  
643   local t = {}  
644   for _,i in ipairs(s:explode("\13+")) do  
645     local k,v = i:match("(.-)=(.*)") -- v may contain = or empty.  
646     if k and v and k ~= "" then  
647       if further_split_keys[k] then  
648         t[k] = v:explode(":")  
649       else  
650         t[k] = v  
651       end  
652     end  
653   end  
654   return t  
655 end  
656
```

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

```
657 local function getobjects(result,figure,f)  
658   return figure:objects()  
659 end  
660  
661 local function convert(result, flusher)  
662   luamplib.flush(result, flusher)  
663   return true -- done  
664 end  
665 luamplib.convert = convert  
666  
667 local function pdf_startfigure(n,llx,lly,urx,ury)  
668   texprint(format("\\"\\mplibstarttoPDF{%.2f}{%.2f}{%.2f}{%.2f}",llx,lly,urx,ury))  
669 end  
670  
671 local function pdf_stopfigure()  
672   texprint("\\"\\mplibstopstoPDF")  
673 end  
674  
tex.tprint with catcode regime -2, as sometimes # gets doubled in the argument of  
pdfliteral.  
675 local function pdf_literalcode(fmt,...) -- table  
676   texprint({"\\"\\mplibtoPDF"},{-2,format(fmt,...)},{""})  
677 end  
678
```

```

679 local function pdf_textfigure(font,size,text,width,height,depth)
680   text = text:gsub(".",function(c)
681     return format("\\"..c.."\\"..c..") -- kerning happens in metapost
682   end)
683   texsprint(format("\\"..font.."\\"..size.."\\"..text.."\\"..width.."\\"..height.."\\"..depth..") )
684 end
685
686 local bend_tolerance = 131/65536
687
688 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
689
690 local function pen_characteristics(object)
691   local t = mplib.pen_info(object)
692   rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
693   divider = sx*sy - rx*ry
694   return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
695 end
696
697 local function concat(px, py) -- no tx, ty here
698   return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
699 end
700
701 local function curved(ith,pth)
702   local d = pth.left_x - ith.right_x
703   if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x - d) <= bend_tolerance then
704     d = pth.left_y - ith.right_y
705     if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y - d) <= bend_tolerance then
706       return false
707     end
708   end
709   return true
710 end
711
712 local function flushnormalpath(path,open)
713   local pth, ith
714   for i=1,#path do
715     pth = path[i]
716     if not ith then
717       pdf_literalcode("%f %f m",pth.x_coord, pth.y_coord)
718     elseif curved(ith, pth) then
719       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)
720     else
721       pdf_literalcode("%f %f l",pth.x_coord, pth.y_coord)
722     end
723     ith = pth
724   end
725   if not open then
726     local one = path[1]
727     if curved(pth,one) then
728       pdf_literalcode("%f %f %f %f %f c", pth.right_x, pth.right_y, one.left_x, one.left_y, one.x_coord, one.y_coord )

```

```

729     else
730       pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
731     end
732   elseif #path == 1 then -- special case .. draw point
733     local one = path[1]
734     pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
735   end
736 end
737
738 local function flushconcatpath(path,open)
739   pdf_literalcode("%f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
740   local pth, ith
741   for i=1,#path do
742     pth = path[i]
743     if not ith then
744       pdf_literalcode("%f %f m",concat(pth.x_coord, pth.y_coord))
745     elseif curved(ith,pth) then
746       local a, b = concat(ith.right_x,ith.right_y)
747       local c, d = concat(pth.left_x, pth.left_y)
748       pdf_literalcode("%f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
749     else
750       pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
751     end
752     ith = pth
753   end
754   if not open then
755     local one = path[1]
756     if curved(pth,one) then
757       local a, b = concat(pth.right_x, pth.right_y)
758       local c, d = concat(one.left_x, one.left_y)
759       pdf_literalcode("%f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
760     else
761       pdf_literalcode("%f %f l",concat(one.x_coord, one.y_coord))
762     end
763   elseif #path == 1 then -- special case .. draw point
764     local one = path[1]
765     pdf_literalcode("%f %f l",concat(one.x_coord, one.y_coord))
766   end
767 end
768

dvipdfmx is supported, though nobody seems to use it.

769 local pdfoutput = tonumber(texget("outputmode")) or tonumber(texget("pdfoutput"))
770 local pdfmode = pdfoutput > 0
771
772 local function start_pdf_code()
773   if pdfmode then
774     pdf_literalcode("q")
775   else
776     texprint("\special{pdf:bcontent}") -- dvipdfmx

```

```

777   end
778 end
779 local function stop_pdf_code()
780   if pdfmode then
781     pdf_literalcode("Q")
782   else
783     texsprint("\\special{pdf:econtent}") -- dvipdfmx
784   end
785 end
786

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

787 local function put_tex_boxes (object,prescript)
788   local box = prescript.mplibtexboxid
789   local n,tw,th = box[1],tonumber(box[2]),tonumber(box[3])
790   if n and tw and th then
791     local op = object.path
792     local first, second, fourth = op[1], op[2], op[4]
793     local tx, ty = first.x_coord, first.y_coord
794     local sx, rx, ry, sy = 1, 0, 0, 1
795     if tw ~= 0 then
796       sx = (second.x_coord - tx)/tw
797       rx = (second.y_coord - ty)/tw
798       if sx == 0 then sx = 0.00001 end
799     end
800     if th ~= 0 then
801       sy = (fourth.y_coord - ty)/th
802       ry = (fourth.x_coord - tx)/th
803       if sy == 0 then sy = 0.00001 end
804     end
805     start_pdf_code()
806     pdf_literalcode("%f %f %f %f %f cm",sx,rx,ry,sy,tx,ty)
807     texsprint(format("\\mpplibputtextbox{%i}",n))
808     stop_pdf_code()
809   end
810 end
811

```

### Colors and Transparency

```

812 local pdf_objs = {}
813 local token, getpageres, setpageres = newtoken or token
814 local pgf = { bye = "pgfutil@everybye", extgs = "pgf@sys@addpdfresource@extgs@plain" }
815
816 if pdfmode then -- repect luaotfload-colors
817   getpageres = pdf.getpageresources or function() return pdf.pageresources end
818   setpageres = pdf.setpageresources or function(s) pdf.pageresources = s end
819 else
820   texsprint("\\special{pdf:obj @MPlibTr<>}",
821           "\\special{pdf:obj @MPlibSh<>}")
822 end

```

```

823
824 local function update_pdfobjs (os)
825   local on = pdf_objs[os]
826   if on then
827     return on,false
828   end
829   if pdfmode then
830     on = pdf.immediateobj(os)
831   else
832     on = pdf_objs.cnt or 0
833     pdf_objs.cnt = on + 1
834   end
835   pdf_objs[os] = on
836   return on,true
837 end
838
839 local transparency_modes = { [0] = "Normal",
840   "Normal",      "Multiply",      "Screen",      "Overlay",
841   "SoftLight",    "HardLight",    "ColorDodge",   "ColorBurn",
842   "Darken",       "Lighten",       "Difference",  "Exclusion",
843   "Hue",          "Saturation",   "Color",        "Luminosity",
844   "Compatible",
845 }
846
847 local function update_tr_res(res,mode,opaq)
848   local os = format("<</BM /%s/ca %.3f/CA %.3f/AIS false>>", mode, opaq, opaq)
849   local on, new = update_pdfobjs(os)
850   if new then
851     if pdfmode then
852       res = format("%s/MPlibTr%i %i 0 R",res,on,on)
853     else
854       if pgf.loaded then
855         texsprint(format("\\\csname %s\\\\endcsname{/MPlibTr%i%s}", pgf.extgs, on, os))
856       else
857         texsprint(format("\\special{pdf:put @MPlibTr<</MPlibTr%i%s>>}",on,os))
858       end
859     end
860   end
861   return res,on
862 end
863
864 local function tr_pdf_pageresources(mode,opaq)
865   if token and pgf.bye and not pgf.loaded then
866     pgf.loaded = token.create(pgf.bye).cmdname == "assign_toks"
867     pgf.bye = pgf.loaded and pgf.bye
868   end
869   local res, on_on, off_on = "", nil, nil
870   res, off_on = update_tr_res(res, "Normal", 1)
871   res, on_on = update_tr_res(res, mode, opaq)
872   if pdfmode then

```

```

873     if res ~= "" then
874         if pgf.loaded then
875             texprint(format("\\"csname %s\\endcsname{%"s}", pgf.extgs, res))
876         else
877             local tpr, n = getpageres() or "", 0
878             tpr, n = tpr:gsub("/ExtGState<<", "%1"..res)
879             if n == 0 then
880                 tpr = format("%s/ExtGState<<%s>>", tpr, res)
881             end
882             setpageres(tpr)
883         end
884     end
885     else
886         if not pgf.loaded then
887             texprint(format("\\"special{pdf:put @resources<</ExtGState @MplibTr>>}"))
888         end
889     end
890     return on_on, off_on
891 end
892

```

Shading with metafun format. (maybe legacy way)

```

893 local shading_res
894
895 local function shading_initialize ()
896     shading_res = {}
897     if pdfmode and luatexbase.callbacktypes.finish_pdffile then -- ltluatex
898         local shading_obj = pdf.reserveobj()
899         setpageres(format("%s/Shading %i 0 R",getpageres() or "",shading_obj))
900         luatexbase.add_to_callback("finish_pdffile", function()
901             pdf.immediateobj(shading_obj,format("<<%s>>",tableconcat(shading_res)))
902         end, "luamplib.finish_pdffile")
903         pdf_objs.finishpdf = true
904     end
905 end
906
907 local function sh_pdffpageresources(shtype,domain,colorspace,colora,colorb,coordinates)
908     if not shading_res then shading_initialize() end
909     local os = format("<</FunctionType 2/Domain [ %s ]/C0 [ %s ]/C1 [ %s ]/N 1>>",
910                         domain, colora, colorb)
911     local funcobj = pdfmode and format("%i 0 R",update_pdfobjs(os)) or os
912     os = format("<</ShadingType %i/ColorSpace /%s/Function %s/Coords [ %s ]/Extend [ true true ]/AntiAlias true>>",
913                         shtype, colorspace, funcobj, coordinates)
914     local on, new = update_pdfobjs(os)
915     if pdfmode then
916         if new then
917             local res = format("/MplibSh%i %i 0 R", on, on)
918             if pdf_objs.finishpdf then
919                 shading_res[#shading_res+1] = res
920             else

```

```

921     local pageres = getpageres() or ""
922     if not pageres:find("/Shading<<.*>>") then
923         pageres = pageres.." /Shading<<>>"
924     end
925     pageres = pageres:gsub("/Shading<<","%1..res")
926     setpageres(pageres)
927     end
928   end
929 else
930   if new then
931     texsprint(format("\\"\\special{pdf:put @MPlibSh<</MPlibSh%i%s>>}",on,os))
932   end
933   texsprint(format("\\"\\special{pdf:put @resources<</Shading @MPlibSh>>}"))
934 end
935 return on
936 end
937
938 local function color_normalize(ca,cb)
939   if #cb == 1 then
940     if #ca == 4 then
941       cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
942     else -- #ca = 3
943       cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
944     end
945   elseif #cb == 3 then -- #ca == 4
946     cb[1], cb[2], cb[3], cb[4] = 1-cb[1], 1-cb[2], 1-cb[3], 0
947   end
948 end
949
950 local prev_override_color
951
952 local function do_preobj_color(object,script)
953   transparency
954   local opaq = script and script.tr_transparency
955   local tron_no, troff_no
956   if opaq then
957     local mode = script.tr_alternative or 1
958     mode = transparancy_modes[tonumber(mode)]
959     tron_no, troff_no = tr_pdf_pageresources(mode,opaq)
960     pdf_literalcode("/MPlibTr%i gs",tron_no)
961   end
962   color
963   local override = script and script.MPlibOverrideColor
964   if override then
965     if pdfmode then
966       pdf_literalcode(override)
967       override = nil
968     else
969       texsprint(format("\\"\\special{color push %s}",override))

```

```

968     prev_override_color = override
969   end
970 else
971   local cs = object.color
972   if cs and #cs > 0 then
973     pdf_literalcode(luamplib.colorconverter(cs))
974     prev_override_color = nil
975   elseif not pdfmode then
976     override = prev_override_color
977     if override then
978       texprint(format("\special{color push %s}",override))
979     end
980   end
981 end
982
983 shading
984 local sh_type = prescript and prescript.sh_type
985 if sh_type then
986   local domain  = prescript.sh_domain
987   local centera = prescript.sh_center_a:explode()
988   local centerb = prescript.sh_center_b:explode()
989   for _,t in pairs({centera,centerb}) do
990     for i,v in ipairs(t) do
991       t[i] = format("%f",v)
992     end
993   end
994   centera = tableconcat(centera," ")
995   centerb = tableconcat(centerb," ")
996   local colora  = prescript.sh_color_a or {0};
997   local colorb  = prescript.sh_color_b or {1};
998   for _,t in pairs({colora,colorb}) do
999     for i,v in ipairs(t) do
1000       t[i] = format("%.3f",v)
1001     end
1002   end
1003   if #colora > #colorb then
1004     color_normalize(colora,colorb)
1005   elseif #colorb > #colora then
1006     color_normalize(colorb,colora)
1007   end
1008   local colorspace
1009   if #colorb == 1 then colorspace = "DeviceGray"
1010   elseif #colorb == 3 then colorspace = "DeviceRGB"
1011   elseif #colorb == 4 then colorspace = "DeviceCMYK"
1012   else  return troff_no,override
1013   end
1014   colora = tableconcat(colora, " ")
1015   colorb = tableconcat(colorb, " ")
1016   local shade_no
1017   if sh_type == "linear" then

```

```

1016     local coordinates = tableconcat({centera,centerb}, " ")
1017     shade_no = sh_pdpageresources(2, domain, colorspace, colora, colorb, coordinates)
1018 elseif sh_type == "circular" then
1019     local radiusa = format("%f",prescript.sh_radius_a)
1020     local radiusb = format("%f",prescript.sh_radius_b)
1021     local coordinates = tableconcat({centera,radiusa,centerb,radiusb}, " ")
1022     shade_no = sh_pdpageresources(3, domain, colorspace, colora, colorb, coordinates)
1023 end
1024 pdf_literalcode("q /Pattern cs")
1025 return troff_no,override,shade_no
1026 end
1027 return troff_no,override
1028 end
1029
1030 local function do_postobj_color(tr,over,sh)
1031 if sh then
1032   pdf_literalcode("W n /MplibSh%$ sh Q",sh)
1033 end
1034 if over then
1035   texprint("\\special{color pop}")
1036 end
1037 if tr then
1038   pdf_literalcode("/MplibTr%$ gs",tr)
1039 end
1040 end
1041
```

Finally, flush figures by inserting PDF literals.

```

1042 local function flush(result,flusher)
1043 if result then
1044   local figures = result.fig
1045   if figures then
1046     for f=1, #figures do
1047       info("flushing figure %s",f)
1048       local figure = figures[f]
1049       local objects = getobjects(result,figure,f)
1050       local fignum = tonumber(figure:filename():match("(%d)+$") or figure:charcode() or 0)
1051       local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
1052       local bbox = figure:boundingbox()
1053       local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
1054       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()
```

```

1055 else
```

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

```

1056      if tex_code_pre_mplib[f] then
1057          texspprint(tex_code_pre_mplib[f])
1058      end
1059      local TeX_code_bot = {}
1060      pdf_startfigure(fignum,l1x,l1y,urx,ury)
1061      start_pdf_code()
1062      if objects then
1063          local savedpath = nil
1064          local savedhtap = nil
1065          for o=1,#objects do
1066              local object      = objects[o]
1067              local objecttype = object.type

```

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

```

1068      local prescription = object.prescription
1069      prescription = prescription and script2table(prescription) -- prescription is now a table
1070      local tr_opaq,cr_over,shade_no = do_preobj_color(object,prescription)
1071      if prescription and prescription.mplibtexboxid then
1072          put_tex_boxes(object,prescription)
1073      elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then --skip
1074      elseif objecttype == "start_clip" then
1075          local evenodd = not object.istext and object.postscript == "evenodd"
1076          start_pdf_code()
1077          flushnormalpath(object.path,false)
1078          pdf_literalcode(evenodd and "%W* n" or "%W n")
1079      elseif objecttype == "stop_clip" then
1080          stop_pdf_code()
1081          miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
1082      elseif objecttype == "special" then

```

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

```

1083      if prescription and prescription.postmplibverbtex then
1084          TeX_code_bot[#TeX_code_bot+1] = prescription.postmplibverbtex
1085      end
1086      elseif objecttype == "text" then
1087          local ot = object.transform -- 3,4,5,6,1,2
1088          start_pdf_code()
1089          pdf_literalcode("%f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
1090          pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
1091          stop_pdf_code()
1092      else
1093          local evenodd, collect, both = false, false, false
1094          local postscript = object.postscript
1095          if not object.istext then
1096              if postscript == "evenodd" then
1097                  evenodd = true
1098              elseif postscript == "collect" then

```

```

1099         collect = true
1100     elseif postscript == "both" then
1101         both = true
1102     elseif postscript == "eoboth" then
1103         evenodd = true
1104         both    = true
1105     end
1106 end
1107 if collect then
1108     if not savedpath then
1109         savedpath = { object.path or false }
1110         savedhtap = { object.htap or false }
1111     else
1112         savedpath[#savedpath+1] = object.path or false
1113         savedhtap[#savedhtap+1] = object.htap or false
1114     end
1115 else
1116     local ml = object.miterlimit
1117     if ml and ml ~= miterlimit then
1118         miterlimit = ml
1119         pdf_literalcode("%f M",ml)
1120     end
1121     local lj = object.linejoin
1122     if lj and lj ~= linejoin then
1123         linejoin = lj
1124         pdf_literalcode("%i j",lj)
1125     end
1126     local lc = object.linecap
1127     if lc and lc ~= linecap then
1128         linecap = lc
1129         pdf_literalcode("%i J",lc)
1130     end
1131     local dl = object.dash
1132     if dl then
1133         local d = format("[%s] %f d",tableconcat(dl.dashes or {}," "),dl.offset)
1134         if d ~= dashed then
1135             dashed = d
1136             pdf_literalcode(dashed)
1137         end
1138     elseif dashed then
1139         pdf_literalcode("[] 0 d")
1140         dashed = false
1141     end
1142     local path = object.path
1143     local transformed, penwidth = false, 1
1144     local open = path and path[1].left_type and path[#path].right_type
1145     local pen = object.pen
1146     if pen then
1147         if pen.type == 'elliptical' then
1148             transformed, penwidth = pen_characteristics(object) -- boolean, value

```

```

1149           pdf_literalcode("%f w",penwidth)
1150           if objecttype == 'fill' then
1151               objecttype = 'both'
1152           end
1153           else -- calculated by mpplib itself
1154               objecttype = 'fill'
1155           end
1156       end
1157       if transformed then
1158           start_pdf_code()
1159       end
1160       if path then
1161           if savedpath then
1162               for i=1,#savedpath do
1163                   local path = savedpath[i]
1164                   if transformed then
1165                       flushconcatpath(path,open)
1166                   else
1167                       flushnormalpath(path,open)
1168                   end
1169               end
1170               savedpath = nil
1171           end
1172           if transformed then
1173               flushconcatpath(path,open)
1174           else
1175               flushnormalpath(path,open)
1176           end

```

Change from ConTeXt general: there was color stuffs.

```

1177           if not shade_no then -- conflict with shading
1178               if objecttype == "fill" then
1179                   pdf_literalcode(evenodd and "h f*" or "h f")
1180               elseif objecttype == "outline" then
1181                   if both then
1182                       pdf_literalcode(evenodd and "h B*" or "h B")
1183                   else
1184                       pdf_literalcode(open and "S" or "h S")
1185                   end
1186               elseif objecttype == "both" then
1187                   pdf_literalcode(evenodd and "h B*" or "h B")
1188               end
1189           end
1190           if transformed then
1191               stop_pdf_code()
1192           end
1193           local path = object.htap
1194           if path then
1195               if transformed then

```

```

1197         start_pdf_code()
1198     end
1199     if savedhtap then
1200         for i=1,#savedhtap do
1201             local path = savedhtap[i]
1202             if transformed then
1203                 flushconcatpath(path,open)
1204             else
1205                 flushnormalpath(path,open)
1206             end
1207         end
1208         savedhtap = nil
1209         evenodd  = true
1210     end
1211     if transformed then
1212         flushconcatpath(path,open)
1213     else
1214         flushnormalpath(path,open)
1215     end
1216     if objecttype == "fill" then
1217         pdf_literalcode(evenodd and "h f*" or "h f")
1218     elseif objecttype == "outline" then
1219         pdf_literalcode(open and "S" or "h S")
1220     elseif objecttype == "both" then
1221         pdf_literalcode(evenodd and "h B*" or "h B")
1222     end
1223     if transformed then
1224         stop_pdf_code()
1225     end
1226     end
1227 end
1228 end

```

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

```

1229         do_postobj_color(tr_opaq,cr_over,shade_no)
1230     end
1231 end
1232 stop_pdf_code()
1233 pdf_stopfigure()
1234 if #TeX_code_bot > 0 then texsprint(TeX_code_bot) end
1235 end
1236 end
1237 end
1238 end
1239 end
1240 luamplib.flush = flush
1241
1242 local function colorconverter(cr)
1243     local n = #cr
1244     if n == 4 then

```

```

1245     local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
1246     return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
1247 elseif n == 3 then
1248     local r, g, b = cr[1], cr[2], cr[3]
1249     return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
1250 else
1251     local s = cr[1]
1252     return format("%.3f g %.3f G",s,s), "0 g 0 G"
1253 end
1254 end
1255 luamplib.colorconverter = colorconverter

```

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

First we need to load some packages.

```

1256 \bgroup\expandafter\expandafter\expandafter\egroup
1257 \expandafter\ifx\csname selectfont\endcsname\relax
1258   \input ltluatex
1259 \else
1260   \NeedsTeXFormat{LaTeXe}
1261   \ProvidesPackage{luamplib}
1262   [2020/02/24 v2.20.5 mpilib package for LuaTeX]
1263   \ifx\newluafunction@\undefined
1264     \input ltluatex
1265   \fi
1266 \fi

```

Loading of lua code.

```
1267 \directlua{require("luamplib")}
```

Support older engine. Seems we don't need it, but no harm.

```

1268 \ifx\pdfoutput\undefined
1269   \let\pdfoutput\outputmode
1270   \protected\def\pdfliteral{\pdfextension literal}
1271 \fi

```

Unfortuantely 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.

```

1272 \ifx\pdfliteral\undefined
1273   \protected\def\pdfliteral{\pdfextension literal}
1274 \fi

```

Set the format for metapost.

```
1275 \def\mplibsetformat#1{\directlua{luamplib.setformat("#1")}}
```

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

```

1276 \ifnum\pdfoutput>0
1277   \let\mplibtoPDF\pdfliteral
1278 \else

```

```

1279 \def\mplibtoPDF#1{\special{pdf:literal direct #1}}
1280 \ifcsname PackageWarning\endcsname
1281   \PackageWarning{luamplib}{take dvipdfmx path, no support for other dvi tools currently.}
1282 \else
1283   \write128{}
1284   \write128{luamplib Warning: take dvipdfmx path, no support for other dvi tools currently.}
1285   \write128{}
1286 \fi
1287 \fi

      Make mplibcode typesetted always in horizontal mode.

1288 \def\mplibforcehmode{\let\prependtomplibbox\leavevmode}
1289 \def\mplibnoforcehmode{\let\prependtomplibbox\relax}
1290 \mplibnoforcehmode

      Catcode. We want to allow comment sign in mplibcode.

1291 \def\mplibsetupcatcodes%
1292   %catcode`#=12 %catcode`\}=12
1293   \catcode`\#=12 \catcode`\^=12 \catcode`\~=12 \catcode`\_=12
1294   \catcode`\&=12 \catcode`\$=12 \catcode`\%=12 \catcode`\^^M=12
1295 }

      Make btx...etex box zero-metric.

1296 \def\mplibputtextbox#1{\vbox to 0pt{\vss\hbox to 0pt{\raise\dp#1\copy#1\hss}}}

      The Plain-specific stuff.

1297 \bgroup\expandafter\expandafter\expandafter\egroup
1298 \expandafter\ifx\csname selectfont\endcsname\relax
1299 \def\mplibcode{%
1300   \begingroup
1301   \begingroup
1302   \mplibsetupcatcodes
1303   \mplibdocode
1304 }
1305 \long\def\mplibdocode#1\endmplibcode{%
1306   \endgroup
1307   \directlua{luamplib.process_mplibcode([==[\unexpanded{#1}]]==])}%
1308   \endgroup
1309 }
1310 \else

      The LATEX-specific part: a new environment.

1311 \newenvironment{mplibcode}{%
1312   \mplibmptoks{}\ltxdomplibcode
1313 }{%
1314 \def\ltxdomplibcode{%
1315   \begingroup
1316   \mplibsetupcatcodes
1317   \ltxdomplibcodeindeed
1318 }
1319 \def\mplib@mplibcode{mplibcode}
1320 \long\def\ltxdomplibcodeindeed#1\end#2{%

```

```

1321 \endgroup
1322 \mplibtmptoks\expandafter{\the\mplibtmptoks#1}%
1323 \def\mplibtemp@a{#2}%
1324 \ifx\mpplib@mplicode\mplibtemp@a
1325   \directlua{luamplib.process_mplicode([==[\the\mplibtmptoks]==])}%
1326   \end{mplicode}%
1327 \else
1328   \mplibtmptoks\expandafter{\the\mplibtmptoks\end{#2}}%
1329   \expandafter\ltxdomplibcode
1330 \fi
1331 }
1332 \fi

User settings.

1333 \def\mpliblegacybehavior#1{\directlua{
1334   local s = string.lower("#1")
1335   if s == "enable" or s == "true" or s == "yes" then
1336     luamplib.legacy_verbatimtex = true
1337   else
1338     luamplib.legacy_verbatimtex = false
1339   end
1340 }}
1341 \def\mplibverbatim#1{\directlua{
1342   local s = string.lower("#1")
1343   if s == "enable" or s == "true" or s == "yes" then
1344     luamplib.verbatiminput = true
1345   else
1346     luamplib.verbatiminput = false
1347   end
1348 }}
1349 \newtoks\mplibtmptoks
\everympplib & \everyendmpplib: macros redefining \everymplibtoks & \everyendmplibtoks
respectively

1350 \newtoks\everymplibtoks
1351 \newtoks\everyendmplibtoks
1352 \protected\def\everympplib{%
1353   \begingroup
1354   \mplibsetupcatcodes
1355   \mplibdoeverympplib
1356 }
1357 \long\def\mplibdoeverympplib#1{%
1358   \endgroup
1359   \everymplibtoks{#1}%
1360 }
1361 \protected\def\everyendmpplib{%
1362   \begingroup
1363   \mplibsetupcatcodes
1364   \mplibdoeveryendmpplib
1365 }

```

```

1366 \long\def\mpplibdoeveryendmpplib#1{%
1367   \endgroup
1368   \everyendmplibtoks{\#1}%
1369 }

```

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.

```

1370 \def\mpdim#1{ mpplibdimen("#1") }
1371 \def\mpcolor#1{\domplibcolor{\#1}}
1372 \def\domplibcolor#1#2{ mpplibcolor("#1{\#2}") }

```

MPLib's number system. Now binary has gone away.

```

1373 \def\mplibnumbersystem#1{\directlua{
1374   local t = "#1"
1375   if t == "binary" then t = "decimal" end
1376   luamplib.numbersystem = t
1377 }}

```

Settings for .mp cache files.

```

1378 \def\mplibmakenocache#1{\mpplibdomakenocache #1,*,*}
1379 \def\mpplibdomakenocache#1,{%
1380   \ifx\empty\empty
1381     \expandafter\mpplibdomakenocache
1382   \else
1383     \ifx*#1\else
1384       \directlua{\luamplib.noneedtoreplace["#1.mp"]=true}%
1385       \expandafter\expandafter\expandafter\mpplibdomakenocache
1386     \fi
1387   \fi
1388 }
1389 \def\mplibcancelnocache#1{\mpplibdocancelnocache #1,*,*}
1390 \def\mpplibdocancelnocache#1,{%
1391   \ifx\empty\empty
1392     \expandafter\mpplibdocancelnocache
1393   \else
1394     \ifx*#1\else
1395       \directlua{\luamplib.noneedtoreplace["#1.mp"]=false}%
1396       \expandafter\expandafter\expandafter\mpplibdocancelnocache
1397     \fi
1398   \fi
1399 }
1400 \def\mplibcachedir#1{\directlua{\luamplib.getcachedir("\unexpanded{\#1})}}

```

More user settings.

```

1401 \def\mplibtexttextlabel#1{\directlua{
1402   local s = string.lower("#1")
1403   if s == "enable" or s == "true" or s == "yes" then
1404     luamplib.texttextlabel = true
1405   else
1406     luamplib.texttextlabel = false

```

```

1407     end
1408   }}
1409 \def\mplibcodeinherit#1{\directlua{
1410   local s = string.lower("#1")
1411   if s == "enable" or s == "true" or s == "yes" then
1412     luamplib.codeinherit = true
1413   else
1414     luamplib.codeinherit = false
1415   end
1416 }
1417 \def\mplibglobaltexttext#1{\directlua{
1418   local s = string.lower("#1")
1419   if s == "enable" or s == "true" or s == "yes" then
1420     luamplib.globaltexttext = true
1421   else
1422     luamplib.globaltexttext = false
1423   end
1424 }

```

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

```
1425 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
```

We encapsulate the litterals.

```

1426 \def\mplibstarttoPDF#1#2#3#4{%
1427   \prependtomplibbox
1428   \hbox\bgroup
1429   \xdef\MPllx{\#1}\xdef\MPlly{\#2}%
1430   \xdef\MPurx{\#3}\xdef\MPury{\#4}%
1431   \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
1432   \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
1433   \parskip0pt%
1434   \leftskip0pt%
1435   \parindent0pt%
1436   \everypar{}%
1437   \setbox\mplibscratchbox\vbox\bgroup
1438   \noindent
1439 }
1440 \def\mplibstopoPDF{%
1441   \egroup %
1442   \setbox\mplibscratchbox\hbox %
1443   {\hskip-\MPllx bp%
1444     \raise-\MPlly bp%
1445     \box\mplibscratchbox}%
1446   \setbox\mplibscratchbox\vbox to \MPheight
1447   {\vfill
1448     \hsize\MPwidth
1449     \wd\mplibscratchbox0pt%
1450     \ht\mplibscratchbox0pt%
1451     \dp\mplibscratchbox0pt%
1452     \box\mplibscratchbox}%
1453   \wd\mplibscratchbox\MPwidth

```

```
1454 \ht\mplibscratchbox\MPheight  
1455 \box\mplibscratchbox  
1456 \egroup  
1457 }
```

Text items have a special handler.

```
1458 \def\mplibtexttext#1#2#3#4#5{  
1459 \begingroup  
1460 \setbox\mplibscratchbox\hbox  
1461 {\font\temp=#1 at #2bp%  
1462 \temp  
1463 #3}  
1464 \setbox\mplibscratchbox\hbox  
1465 {\hskip#4 bp%  
1466 \raise#5 bp%  
1467 \box\mplibscratchbox}  
1468 \wd\mplibscratchbox0pt%  
1469 \ht\mplibscratchbox0pt%  
1470 \dp\mplibscratchbox0pt%  
1471 \box\mplibscratchbox  
1472 \endgroup  
1473 }
```

Input luamplib.cfg when it exists.

```
1474 \openin\luamplib=\luamplib.cfg  
1475 \ifeof\luamplib \else  
1476 \closein\luamplib  
1477 \input luamplib.cfg  
1478 \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 allowed.

##### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the General Public License is intended to guarantee that you have the freedom to share and change free software--to make sure that the same freedoms that others enjoyed are also available to you. 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 make a difference by distributing this license in a package.

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 redistribute 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, and you must not restrict them so they cannot give their copies away, either.

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, that person should receive a written notice that says that, in effect, "This program is provided 'as is' without any warranty of any kind, including any implied warranty of merchantability or fitness for a particular purpose." The entire risk as to the quality and performance of the program is yours. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

Finally, any free program is intended eventually to be replaced by a new version that is different from the one that you received. After you receive a new version, you are welcome to provide it to others; receiving copies of the new version is governed by this License. You must also receive the new version in order to use the new features or modifications that it contains.

Most software is distributed in the hope that it will be useful, but without any warranty or promise that it will be. If it fails, the author is not liable for damage caused by it.

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

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

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of the General Public License. The "Program" below refers to any such program or work, and "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing portions of the Program, or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is addressed as "modification".) 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 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.

2. 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 a appropriate copyright notice and disclaimer of warranty; keep intact all notices that refer to this License and to the absence of any warranty; and give all recipient copies 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.

3. 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:

(a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) 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.

(c) If the modified program normally sends command-line arguments that run you must cause those arguments to be accepted correctly, and must cause an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) to be printed when the modified version is run. If it outputs a warning or a note printed in the middle of the distribution system, it is sufficient that it prints the warning or note before it views 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 separated out, then this License does not apply to those sections. (Copyright holders who places the Program in a library may add an explicit geographical distribution limitation excluding those countries where separate 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.)

on the terms of this License, whose permissions for other licenses 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 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.

10. 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 will be 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 of this license, you may choose any version ever published by the Free Software Foundation.

11. 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. One decision will be guided by 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

12. 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 WHO MAY MAKE AVAILABLE SOURCE CODE FOR THE PROGRAM DO SO AS IS AND WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION 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.

13. 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 REDISTRIBUTE 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 THOUGH SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### END OF TERMS AND CONDITIONS

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

If you wish to apply some or all of this License to your own programs, follow the terms and conditions of this License. Otherwise, you will need to add additional terms and conditions to cover your specific circumstances. See Section 7.

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 also include them in any documentation for the program, if they are not already present there.

You may add as many of this license as you wish to each source file to further delineate each part.

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 c' and 'show w' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something else that sounds like 'show c' and 'show w' if 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 one, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

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

signature of Ty Coon, 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 legal 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.