The luamplib package

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

2024/05/01 v2.29.0

Abstract

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

1 Documentation

This packages aims at providing a simple way to typeset directly metapost code in a document with LuaTEX. LuaTEX is built with the lua `mplib` library, that runs metapost code. This package is basically a wrapper (in Lua) for the Lua `mplib` functions and some TEX functions to have the output of the `mplib` functions in the pdf.

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

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

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

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

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

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

  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.)

\mpfig ... \endmpfig Since v2.29 we provide unexpandable TEX macros \mpfig ... \endmpfig and its starred version \mpfig* ... \endmpfig to save typing toil. The first is roughly the same as follows:

```
\begin{mplibcode}[@mpfig]
beginfig(0)
token list declared by \everymplib[@mpfig]
...
token list declared by \everyendmplib[@mpfig]
endfig;
\end{mplibcode}
```

and the starred version is roughly the same as follows:

```
\begin{mplibcode}[@mpfig]
...
\end{mplibcode}
```

In these macros \mpliblegacybehavior{disable} (see below) is forcibly declared. And as both share the same instance name, metapost codes are inherited among them. A simple example:

```
\mpfig* input boxes \endmpfig
\everymplib[@mpfig]{ drawoptions(withcolor .5[red,white]); }
\mpfig circleit.a(btex Box 1 etex); drawboxed(a); \endmpfig
```

The instance name (default: @mpfig) can be changed by redefining \mpfiginstancename, after which a new MPlib instance will start and code inheritance too will begin anew. \let\mpfiginstancename\empty will prevent code inheritance if \mplibcodeinherit{true} (see below) is not declared.[1]

\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
```

[1] As for user setting values, enable, true, yes are identical, and disable, false, no are identical.

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 inpired 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 enviroment (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.

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

\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 and plain TEX v2.22 has added the support for several named MetaPost instances in LATEX mplibcode environment. (And since v2.29 plain TEX users can use this functionality as well.) 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 \mplibglobaltextext.
- When an instance names is set, respective \currentmpinstancename is set.

In parellel 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]{...}
```

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

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

Generally speaking, it is recommended to turn mplibglobaltextext always on, because it has the advantage of reusing metapost pictures among code chunks sharing the same mplib instance. 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 verbatimtex ... 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 nececcsary, before returning their paths to LuaTEX's mplib library. This would make the compilation time longer wastefully, as most .mp files do not contain btex ... etex command. So luamplib provides macros as follows, so that users can give instruction about files that do not require this functionality.

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

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

By default, cache files will be stored in $TEXMFVAR/luamplib_cache or, if it's not available (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.

mplibtexcolor, mplibrgbtexcolor mplibtexcolor is a metapost operator that converts a TEX color expression to a MetaPost color expression. For instance:

```
color col; col := mplibtexcolor "olive!50";
```

The result may vary in its color model (gray/rgb/cmyk) according to the given TEX color. (Spot colors are forced to cmyk model, so this operator is not recommended for spot colors.) Therefore the example shown above would raise a metapost error: `cmykcolor col;` should have been declared. By contrast, mplibrgbtexcolor always returns rgb model expressions.

mplibgraphictext For some amusement, luamplib provides its own metapost operator mplibgraphictext, the effect of which is similar to that of ConTEXt's graphictext. However syntax is somewhat different.

```
mplibgraphictext "Funny"
  fakebold 2.3 scale 3               % fontspec options
  drawcolor .7blue fillcolor "red!50" % color expressions
```

fakebold, scale, drawcolor and fillcolor are optional; default values are 2, 1, "black" and "white" respectively. When color expressions are given as string, they are regarded as xcolor's or l3color's expressions (this is the same with shading colors). All from mplibgraphictext to the end of sentence will compose an anonymous picture, which can be drawn or assigned to a variable. Incidentally, withdrawcolor and withfillcolor are synonyms of drawcolor and fillcolor, hopefully to be compatible with graphictext. N.B. Because luamplib's current implementation is quite different from the ConTEXt's, there are some limitations such that you can't apply shading (gradient colors) to the text.

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       = "2.29.0",
```

```
  date          = "2024/05/01",
  description   = "Lua package to typeset Metapost with LuaTeX's MPLib.",
}

```

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

```
luamplib          = luamplib or { }
local luamplib    = luamplib

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

```

Use our own function for warn/info/err.

```
local function termorlog (target, text, kind)
  if text then
    local mod, write, append = "luamplib", texio.write_nl, texio.write
    kind = kind
        or target == "term" and "Warning (more info in the log)"
        or target == "log" and "Info"
        or target == "term and log" and "Warning"
        or "Error"
    target = kind == "Error" and "term and log" or target
    local t = text:explode"\n+"
    write(target, format("Module %s %s:", mod, kind))
    if #t == 1 then
      append(target, format(" %s", t[1]))
    else
      for _,line in ipairs(t) do
        write(target, line)
      end
      write(target, format("(%s)     ", mod))
    end
    append(target, format(" on input line %s", tex.inputlineno))
    write(target, "")
    if kind == "Error" then error() end
  end
end

local function warn (...) -- beware '%' symbol
  termorlog("term and log", select("#",...) > 1 and format(...) or ...)
end
local function info (...)
  termorlog("log", select("#",...) > 1 and format(...) or ...)
end
local function err (...)
  termorlog("error", select("#",...) > 1 and format(...) or ...)
end

luamplib.showlog  = luamplib.showlog or false

```

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

```
local tableconcat = table.concat
local texsprint   = tex.sprint
```

```
local texgettoks  = tex.gettoks
local texgetbox   = tex.getbox
local texruntoks  = tex.runtoks
```

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

```
  local texscantoks = tex.scantoks


if not texruntoks then
  err("Your LuaTeX version is too old. Please upgrade it to the latest")
end

local is_defined  = token.is_defined
local get_macro   = token.get_macro

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

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

```

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

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

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

```

`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.

```
local luamplibtime = kpse.find_file("luamplib.lua")
```

```
luamplibtime = luamplibtime and lfsattributes(luamplibtime,"modification")

local currenttime = os.time()

local outputdir
if lfstouch then
  for i,v in ipairs{'TEXMFVAR','TEXMF_OUTPUT_DIRECTORY','.','TEXMFOUTPUT'} do
    local var = i == 3 and v or kpse.var_value(v)
    if var and var ~= "" then
      for _,vv in next, var:explode(os.type == "unix" and ":" or ";") do
        local dir = format("%s/%s",vv,"luamplib_cache")
        if not lfsisdir(dir) then
          mk_full_path(dir)
        end
        if is_writable(dir) then
          outputdir = dir
          break
        end
      end
      if outputdir then break end
    end
  end
end
outputdir = outputdir or '.'

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

```

Some basic MetaPost files not necessary to make cache files.

```
local noneedtoreplace = {
  ["boxes.mp"] = true, --  ["format.mp"] = true,
  ["graph.mp"] = true, ["marith.mp"] = true, ["mfplain.mp"] = true,
  ["mpost.mp"] = true, ["plain.mp"] = true, ["rboxes.mp"] = true,
  ["sarith.mp"] = true, ["string.mp"] = true, -- ["TEX.mp"] = true,
  ["metafun.mp"] = true, ["metafun.mpiv"] = true, ["mp-abck.mpiv"] = true,
  ["mp-apos.mpiv"] = true, ["mp-asnc.mpiv"] = true, ["mp-bare.mpiv"] = true,
  ["mp-base.mpiv"] = true, ["mp-blob.mpiv"] = true, ["mp-butt.mpiv"] = true,
  ["mp-char.mpiv"] = true, ["mp-chem.mpiv"] = true, ["mp-core.mpiv"] = true,
  ["mp-crop.mpiv"] = true, ["mp-figs.mpiv"] = true, ["mp-form.mpiv"] = true,
  ["mp-func.mpiv"] = true, ["mp-grap.mpiv"] = true, ["mp-grid.mpiv"] = true,
```

```
  ["mp-grph.mpiv"] = true, ["mp-idea.mpiv"] = true, ["mp-luas.mpiv"] = true,
  ["mp-mlib.mpiv"] = true, ["mp-node.mpiv"] = true, ["mp-page.mpiv"] = true,
  ["mp-shap.mpiv"] = true, ["mp-step.mpiv"] = true, ["mp-text.mpiv"] = true,
  ["mp-tool.mpiv"] = true, ["mp-cont.mpiv"] = true,
}
luamplib.noneedtoreplace = noneedtoreplace

```

format.mp is much complicated, so specially treated.

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

```

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

```
local name_b = "%f[%a_]"
local name_e = "%f[^%a_]"
local btex_etex = name_b.."btex"..name_e.."%s*(.-)%s*"..name_b.."etex"..name_e
local verbatimtex_etex = name_b.."verbatimtex"..name_e.."%s*(.-)%s*"..name_b.."etex"..name_e

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

  if name == "format.mp" then return replaceformatmp(file,newfile,ofmodify) end

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

```

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

```
  local count,cnt = 0,0
  data, cnt = data:gsub(btex_etex, "btex %1 etex ") -- space
  count = count + cnt
  data, cnt = data:gsub(verbatimtex_etex, "verbatimtex %1 etex;") -- semicolon
  count = count + cnt

  if count == 0 then
    noneedtoreplace[name] = true
    fh = ioopen(newfile,"w");
    if fh then
      fh:close()
      lfstouch(newfile,currenttime,ofmodify)
    end
    return file
  end

  fh = ioopen(newfile,"w")
  if not fh then return file end
  fh:write(data); fh:close()
  lfstouch(newfile,currenttime,ofmodify)
  return newfile
end

```

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.

```
local mpkpse
do
  local exe = 0
  while arg[exe-1] do
    exe = exe-1
  end
  mpkpse = kpse.new(arg[exe], "mpost")
end

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

local function finder(name, mode, ftype)
  if mode == "w" then
    if name and name ~= "mpout.log" then
      kpse.record_output_file(name) -- recorder
    end
    return name
  else
    ftype = special_ftype[ftype] or ftype
    local file = mpkpse:find_file(name,ftype)
    if file then
      if lfstouch and ftype == "mp" and not noneedtoreplace[name] then
        file = replaceinputmpfile(name,file)
```

```
250       end
251     else
252       file = mpkpse:find_file(name, name:match("%a+$"))
253     end
254     if file then
255       kpse.record_input_file(file) -- recorder
256     end
257     return file
258   end
259 end
260 luamplib.finder = finder
261
```

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.)

```
262 local preamble = [[
263   boolean mplib ; mplib := true ;
264   let dump = endinput ;
265   let normalfontsize = fontsize;
266   input %s ;
267 ]]
268
```

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

```
269 local currentformat = "plain"
270 local function setformat (name)
271   currentformat = name
272 end
273 luamplib.setformat = setformat
274
```

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

```
275 luamplib.codeinherit = false
276 local mplibinstances = {}
277 local has_instancename = false
278
279 local function reporterror (result, prevlog)
280   if not result then
281     err("no result object returned")
282   else
283     local t, e, l = result.term, result.error, result.log
```

log has more information than term, so log first (2021/08/02)

```
284     local log = l or t or "no-term"
285     log = log:gsub("%(Please type a command or say 'end'%)",""):gsub("\n+","\n")
286     if result.status > 0 then
287       local first = log:match"(.-\n! .-)\n! "
288       if first then
289         termorlog("term", first)
290         termorlog("log", log, "Warning")
291       else
292         warn(log)
293       end
294       if result.status > 1 then
295         err(e or "see above messages")
```

```
      end
    elseif prevlog then
      log = prevlog..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 an info, even if output has no figure.

```
      local show = log:match"\n>>? .+"
      if show then
        termorlog("term", show, "Info (more info in the log)")
        info(log)
      elseif luamplib.showlog and log:find"%g" then
        info(log)
      end
    end
    return log
  end
end

local function luamplibload (name)
  local mpx = mplib.new {
    ini_version = true,
    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.

```
    make_text   = luamplib.maketext,
    run_script  = luamplib.runscript,
    math_mode   = luamplib.numbersystem,
    job_name    = tex.jobname,
    random_seed = math.random(4095),
    extensions  = 1,
  }
```

Append our own MetaPost preamble to the preamble above.

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

```

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

```
local function process (data, instancename)
```

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
```

```
  local currfmt
  if instancename and instancename ~= ”” then
    currfmt = instancename
    has_instancename = true
  else
    currfmt = tableconcat{
      currentformat,
      luamplib.numbersystem or ”scaled”,
      tostring(luamplib.textextlabel),
      tostring(luamplib.legacy_verbatimtex),
    }
    has_instancename = false
  end
  local mpx = mplibinstances[currfmt]
  local standalone = not (has_instancename or luamplib.codeinherit)
  if mpx and standalone then
    mpx:finish()
  end
  local log = ””
  if standalone or not mpx then
    mpx, _, log = luamplibload(currentformat)
    mplibinstances[currfmt] = mpx
  end
  local converted, result = false, {}
  if mpx and data then
    result = mpx:execute(data)
    local log = reporterror(result, log)
    if log then
      if result.fig then
        converted = luamplib.convert(result)
      else
        info”No figure output. Maybe no beginfig/endfig”
      end
    end
  else
    err”Mem file unloadable. Maybe generated with a different version of mplib?”
  end
  return converted, result
end

```

dvipdfmx is supported, though nobody seems to use it.

```
local pdfmode = tex.outputmode > 0
```

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

```
local catlatex = luatexbase.registernumber(”catcodetable@latex”)
local catat11  = luatexbase.registernumber(”catcodetable@atletter”)

```

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
```

```
local function run_tex_code (str, cat)
  texruntoks(function() texsprint(cat or catlatex, str) end)
end

```

Prepare textext 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.

```
local texboxes = { globalid = 0, localid = 4096 }
```

For conversion of sp to bp.

```
local factor = 65536*(7227/7200)

local textext_fmt = 'image(addto currentpicture doublepath unitsquare \z
xscaled %f yscaled %f shifted (0,-%f) \z
withprescript "mplibtexboxid=%i:%f:%f")'

local function process_tex_text (str)
  if str then
    local global = (has_instancename or luamplib.globaltextext or luamplib.codeinherit)
                   and "\\global" or ""
    local tex_box_id
    if global == "" then
      tex_box_id = texboxes.localid + 1
      texboxes.localid = tex_box_id
    else
      local boxid = texboxes.globalid + 1
      texboxes.globalid = boxid
      run_tex_code(format(
        [[\expandafter\newbox\csname luamplib.box.%s\endcsname]], boxid))
      tex_box_id = tex.getcount'allocationnumber'
    end
    run_tex_code(format("%s\\setbox%i\\hbox{%s}", global, tex_box_id, str))
    local box = texgetbox(tex_box_id)
    local wd  = box.width  / factor
    local ht  = box.height / factor
    local dp  = box.depth  / factor
    return textext_fmt:format(wd, ht+dp, dp, tex_box_id, wd, ht+dp)
  end
  return ""
end

```

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.

```
local mplibcolorfmt = {
  xcolor = tableconcat{
    [[\begingroup\let\XC@mcolor\relax]],
    [[\def\set@color{\global\mplibtmptoks\expandafter{\current@color}}]],
    [[\color%s\endgroup]],
  },
  l3color = tableconcat{
    [[\begingroup\def\__color_select:N#1{\expandafter\__color_select:nn#1}]],
    [[\def\__color_backend_select:nn#1#2{\global\mplibtmptoks{#1 #2}}]],
    [[\def\__kernel_backend_literal:e#1{\global\mplibtmptoks\expandafter{\expanded{#1}}}]],
    [[\color_select:n%s\endgroup]],
  },
}

local colfmt = is_defined'color_select:n' and "l3color" or "xcolor"
if colfmt == "l3color" then
  run_tex_code{
    "\\newcatcodetable\\luamplibcctabexplat",
    "\\begingroup",
    "\\catcode`@=11 ",
    "\\catcode`_=11 ",
    "\\catcode`:=11 ",
    "\\savecatcodetable\\luamplibcctabexplat",
    "\\endgroup",
  }
end
local ccexplat = luatexbase.registernumber"luamplibcctabexplat"

local function process_color (str, kind)
  if str then
    if not str:find("%b{}") then
      str = format("{%s}",str)
    end
    local myfmt = mplibcolorfmt[colfmt]
    if colfmt == "l3color" and is_defined"color" then
      if str:find("%b[]") then
        myfmt = mplibcolorfmt.xcolor
      else
        for _,v in ipairs(str:match"{(.+)}":explode"!") do
          if not v:find"^%s*%d+%s*$" then
            local pp = get_macro(format("l__color_named_%s_prop",v))
            if not pp or pp == "" then
              myfmt = mplibcolorfmt.xcolor
              break
            end
          end
        end
      end
    end
    if myfmt == mplibcolorfmt.l3color and (kind == "fill" or kind == "draw") then return str end
    run_tex_code(myfmt:format(str), ccexplat or catat11)
```

```
    local t = texgettoks"mplibtmptoks"
    if not pdfmode and not t:find"^pdf" then
      t = t:gsub("%a+ (.+)","pdf:bc [%1]")
    end
    if kind then return t end
    return format('1 withprescript "MPlibOverrideColor=%s"', t)
  end
  return ""
end

local function colorsplit (res)
  local t, tt = { }, res:gsub("[%[%]]",""):explode()
  local be = tt[1]:find"^%d" and 1 or 2
  for i=be, #tt do
    if tt[i]:find"^%a" then break end
    t[#t+1] = tt[i]
  end
  return t
end

luamplib.outlinecolor = function (str, filldraw)
  local nn = filldraw == "fill" and 'fn:=' or 'dn:='
  local cc = filldraw == "fill" and 'fc:=' or 'dc:='
  local res = process_color(str, filldraw)
  if res:match"{(.+)}" == str then
    return format('%s"n"; %s"%s";', nn,cc,str)
  end
  local t = colorsplit(res)
  local md = #t == 1 and 'gray' or #t == 3 and 'rgb' or #t == 4 and 'cmyk'
  return format('%s"nn"; %s"%s}{%s";', nn, cc, md, tableconcat(t,','))
end

luamplib.gettexcolor = function (str, rgb)
  local res = process_color(str, "metapost")
  if res:find" cs " or res:find"@pdf.obj" then
    if not rgb then
      warn("%s is a spot color. Forced to CMYK", str)
    end
    run_tex_code({
      "\\color_export:nnN{",
      str,
      "}{",
      rgb and "space-sep-rgb" or "space-sep-cmyk",
      "}\\mplib_@tempa",
    },ccexplat)
    return get_macro"mplib_@tempa":explode()
  end
  local t = colorsplit(res)
  if #t == 3 or not rgb then return t end
  run_tex_code({ -- force to rgb
    "\\color_export:nnnN{",
    #t == 4 and "cmyk" or "gray",
    "}{",
    tableconcat(t,","),
```

```
    "}{space-sep-rgb}\\mplib_@tempa",
  },ccexplat)
  return get_macro"mplib_@tempa":explode()
end

luamplib.shadecolor = function (str)
  local res = process_color(str, "shade")
  if res:find" cs " or res:find"@pdf.obj" then -- spot color shade: l3 only
```

An example of spot color shading:

```
\documentclass{article}
\usepackage{luamplib}
\mplibsetformat{metafun}
\ExplSyntaxOn
\color_model_new:nnn { pantone3005 }
  { Separation }
  { name = PANTONE~3005~U ,
    alternative-model = cmyk ,
    alternative-values = {1, 0.56, 0, 0}
  }
  \color_set:nnn{spotA}{pantone3005}{1}
  \color_set:nnn{spotB}{pantone3005}{0.6}
\color_model_new:nnn { pantone1215 }
  { Separation }
  { name = PANTONE~1215~U ,
    alternative-model = cmyk ,
    alternative-values = {0, 0.15, 0.51, 0}
  }
  \color_set:nnn{spotC}{pantone1215}{1}
\color_model_new:nnn { pantone2040 }
  { Separation }
  { name = PANTONE~2040~U ,
    alternative-model = cmyk ,
    alternative-values = {0, 0.28, 0.21, 0.04}
  }
  \color_set:nnn{spotD}{pantone2040}{1}
\ExplSyntaxOff
\begin{document}
\begin{mplibcode}
beginfig(1)
  fill unitsquare xyscaled (\mpdim\textwidth,1cm)
       withshademethod "linear"
       withshadevector (0,1)
       withshadestep (
           withshadefraction .5
           withshadecolors ("spotB","spotC")
       )
       withshadestep (
           withshadefraction 1
           withshadecolors ("spotC","spotD")
       )
  ;
endfig;
\end{mplibcode}
```

```
\end{document}
```

```
    run_tex_code({
      [[\color_export:nnN{]], str, [[}{backend}\mplib_@tempa]],
    },ccexplat)
    local name = get_macro'mplib_@tempa':match'{(.-)}{.+}'
    local t, obj = res:explode()
    if pdfmode then
      obj = t[1]:match"^/(.+)"
      if ltx.pdf and ltx.pdf.object_id then
        obj = format("%s 0 R", ltx.pdf.object_id(obj))
      else
        run_tex_code({
          [[\edef\mplib_@tempa{\pdf_object_ref:n{]], obj, "}}",
        },ccexplat)
        obj = get_macro'mplib_@tempa'
      end
    else
      obj = t[2]
    end
    local value = t[3]:match"%[(.-)%]" or t[3]
    return format('(%s) withprescript"mplib_spotcolor=%s:%s"', value,obj,name)
  end
  return colorsplit(res)
end

```

for \mpdim or mplibdimen

```
local function process_dimen (str)
  if str then
    str = str:gsub("{(.+)}","%1")
    run_tex_code(format([[\mplibtmptoks\expandafter{\the\dimexpr %s\relax}]], str))
    return format("begingroup %s endgroup", texgettoks"mplibtmptoks")
  end
  return ""
end

```

Newly introduced method of processing verbatimtex ... etex. This function is used when \mpliblegacybehavior{false} is declared.

```
local function process_verbatimtex_text (str)
  if str then
    run_tex_code(str)
  end
  return ""
end

```

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.

```
local tex_code_pre_mplib = {}
luamplib.figid = 1
luamplib.in_the_fig = false
```

```
local function process_verbatimtex_prefig (str)
  if str then
    tex_code_pre_mplib[luamplib.figid] = str
  end
  return ""
end

local function process_verbatimtex_infig (str)
  if str then
    return format('special "postmplibverbtex=%s";', str)
  end
  return ""
end

local runscript_funcs = {
  luamplibtext    = process_tex_text,
  luamplibcolor   = process_color,
  luamplibdimen   = process_dimen,
  luamplibprefig  = process_verbatimtex_prefig,
  luamplibinfig   = process_verbatimtex_infig,
  luamplibverbtex = process_verbatimtex_text,
}

```

For metafun format. see issue #79.

```
mp = mp or {}
local mp = mp
mp.mf_path_reset = mp.mf_path_reset or function() end
mp.mf_finish_saving_data = mp.mf_finish_saving_data or function() end
mp.report = mp.report or info

```

metafun 2021-03-09 changes crashes luamplib.

```
catcodes = catcodes or {}
local catcodes = catcodes
catcodes.numbers = catcodes.numbers or {}
catcodes.numbers.ctxcatcodes = catcodes.numbers.ctxcatcodes or catlatex
catcodes.numbers.texcatcodes = catcodes.numbers.texcatcodes or catlatex
catcodes.numbers.luacatcodes = catcodes.numbers.luacatcodes or catlatex
catcodes.numbers.notcatcodes = catcodes.numbers.notcatcodes or catlatex
catcodes.numbers.vrbcatcodes = catcodes.numbers.vrbcatcodes or catlatex
catcodes.numbers.prtcatcodes = catcodes.numbers.prtcatcodes or catlatex
catcodes.numbers.txtcatcodes = catcodes.numbers.txtcatcodes or catlatex

```

A function from ConTEXt general.

```
local function mpprint(buffer,...)
  for i=1,select("#",...) do
    local value = select(i,...)
    if value ~= nil then
      local t = type(value)
      if t == "number" then
        buffer[#buffer+1] = format("%.16f",value)
      elseif t == "string" then
```

```
      buffer[#buffer+1] = value
    elseif t == "table" then
      buffer[#buffer+1] = "(" .. tableconcat(value,",") .. ")"
    else -- boolean or whatever
      buffer[#buffer+1] = tostring(value)
    end
  end
end

function luamplib.runscript (code)
  local id, str = code:match("(.-){(.*)}")
  if id and str then
    local f = runscript_funcs[id]
    if f then
      local t = f(str)
      if t then return t end
    end
  end
  local f = loadstring(code)
  if type(f) == "function" then
    local buffer = {}
    function mp.print(...)
      mpprint(buffer,...)
    end
    local res = {f()}
    buffer = tableconcat(buffer)
    if buffer and buffer ~= "" then
      return buffer
    end
    buffer = {}
    mpprint(buffer, table.unpack(res))
    return tableconcat(buffer)
  end
  return ""
end
```

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

```
local function protecttexcontents (str)
  return str:gsub("\\%%", "\0PerCent\0")
            :gsub("%%.-\n", "")
            :gsub("%%.-$",  "")
            :gsub("%zPerCent%z", "\\%%")
            :gsub("%s+", " ")
end

luamplib.legacy_verbatimtex = true

function luamplib.maketext (str, what)
  if str and str ~= "" then
    str = protecttexcontents(str)
    if what == 1 then
      if not str:find("\\documentclass"..name_e) and
         not str:find("\\begin%s*{document}") and
```

```
         not str:find("\\documentstyle"..name_e) and
         not str:find("\\usepackage"..name_e) then
        if luamplib.legacy_verbatimtex then
          if luamplib.in_the_fig then
            return process_verbatimtex_infig(str)
          else
            return process_verbatimtex_prefig(str)
          end
        else
          return process_verbatimtex_text(str)
        end
      end
    else
      return process_tex_text(str)
    end
  end
  return ""
end

```

Our MetaPost preambles

```
local mplibcodepreamble = [[
texscriptmode := 2;
def rawtextext (expr t) = runscript("luamplibtext{"&t&"}") enddef;
def mplibcolor (expr t) = runscript("luamplibcolor{"&t&"}") enddef;
def mplibdimen (expr t) = runscript("luamplibdimen{"&t&"}") enddef;
def VerbatimTeX (expr t) = runscript("luamplibverbtex{"&t&"}") enddef;
def message expr t =
  if string t: runscript("mp.report[=["&t&"]=]") else: errmessage "Not a string" fi
enddef;
if known context_mlib:
  defaultfont := "cmtt10";
  let infont = normalinfont;
  let fontsize = normalfontsize;
  vardef thelabel@#(expr p,z) =
    if string p :
      thelabel@#(p infont defaultfont scaled defaultscale,z)
    else :
      p shifted (z + labeloffset*mfun_laboff@# -
        (mfun_labxf@#*lrcorner p + mfun_labyf@#*ulcorner p +
        (1-mfun_labxf@#-mfun_labyf@#)*llcorner p))
    fi
  enddef;
  def colordecimals primary c =
    if cmykcolor c:
      decimal cyanpart c & ":" & decimal magentapart c & ":" & decimal yellowpart c & ":" & decimal blackpart c
    elseif rgbcolor c:
      decimal redpart c & ":" & decimal greenpart c & ":" & decimal bluepart c
    elseif string c:
      colordecimals resolvedcolor(c)
    else:
      decimal c
    fi
  enddef;
  def resolvedcolor(expr s) =
```

```
    runscript("return luamplib.shadecolor('"& s &"')")
  enddef;
else:
  vardef textext@# (text t) = rawtextext (t) enddef;
fi
def externalfigure primary filename =
  draw rawtextext("\includegraphics{"& filename &"}")
enddef;
def TEX = textext enddef;
def mplibtexcolor primary c =
  runscript("return luamplib.gettexcolor('"& c &"')")
enddef;
def mplibrgbtexcolor primary c =
  runscript("return luamplib.gettexcolor('"& c &"','rgb')")
enddef;
def mplibgraphictext primary t =
  begingroup;
  mplibgraphictext_ (t)
enddef;
def mplibgraphictext_ (expr t) text rest =
  save fakebold, scale, fillcolor, drawcolor, withfillcolor, withdrawcolor,
    fb, sc, fc, dc, fn, dn, tpic;
  picture tpic; tpic := nullpicture;
  numeric fb, sc; string fc, dc, fn, dn;
  fb:=2; sc:=1; fc:="white"; dc:="black"; fn:=dn:="n";
  def fakebold  primary c = hide(fb:=c;) enddef;
  def scale     primary c = hide(sc:=c;) enddef;
  def fillcolor primary c = hide(
    if string c:
      runscript("return luamplib.outlinecolor('"& c &"','fill')")
    else:
      fn:="nn"; fc:=mpliboutlinecolor_(c);
    fi
    ) enddef;
  def drawcolor primary c = hide(
    if string c:
      runscript("return luamplib.outlinecolor('"& c &"','draw')")
    else:
      dn:="nn"; dc:=mpliboutlinecolor_(c);
    fi
  ) enddef;
  let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
  addto tpic doublepath origin rest; tpic:=nullpicture;
  def fakebold  primary c = enddef;
  def scale     primary c = enddef;
  def fillcolor primary c = enddef;
  def drawcolor primary c = enddef;
  let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
  image(draw rawtextext(
  "{\addfontfeature{FakeBold="& decimal fb &",Scale="& decimal sc &
  "}\csname color_fill:"& fn &"\endcsname{"& fc &
  "}\csname color_stroke:"& dn &"\endcsname{"& dc &
  "}"& t &"}") rest;)
  endgroup;
```

```
enddef;
def mpliboutlinecolor_ (expr c) =
  if color c:
    "rgb}{" & decimal redpart c & "," & decimal greenpart c
      & "," & decimal bluepart c
  elseif cmykcolor c:
    "cmyk}{" & decimal cyanpart c & "," & decimal magentapart c
      & "," & decimal yellowpart c & "," & decimal blackpart c
  else:
    "gray}{" & decimal c
  fi
enddef;
]]
luamplib.mplibcodepreamble = mplibcodepreamble

local legacyverbatimtexpreamble = [[
def specialVerbatimTeX (text t) = runscript("luamplibprefig{"&t&"}") enddef;
def normalVerbatimTeX  (text t) = runscript("luamplibinfig{"&t&"}") enddef;
let VerbatimTeX = specialVerbatimTeX;
extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;"&
  "runscript(" &ditto& "luamplib.in_the_fig=true" &ditto& ");";
extra_endfig := extra_endfig & " let VerbatimTeX = specialVerbatimTeX;"&
  "runscript(" &ditto&
  "if luamplib.in_the_fig then luamplib.figid=luamplib.figid+1 end "&
  "luamplib.in_the_fig=false" &ditto& ");";
]]
luamplib.legacyverbatimtexpreamble = legacyverbatimtexpreamble

local textextlabelpreamble = [[
primarydef s infont f = rawtextext(s) enddef;
def fontsize expr f =
  begingroup
  save size; numeric size;
  size := mplibdimen("1em");
  if size = 0: 10pt else: size fi
  endgroup
enddef;
]]
luamplib.textextlabelpreamble = textextlabelpreamble
```

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

```
luamplib.verbatiminput = false
```

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

```
local function protect_expansion (str)
  if str then
    str = str:gsub("\\","!!!Control!!!")
              :gsub("%%","!!!Comment!!!")
              :gsub("#", "!!!HashSign!!!")
              :gsub("{", "!!!LBrace!!!")
              :gsub("}", "!!!RBrace!!!")
    return format("\\unexpanded{%s}",str)
  end
```

```
end

local function unprotect_expansion (str)
  if str then
    return str:gsub("!!!Control!!!", "\\")
              :gsub("!!!Comment!!!", "%%")
              :gsub("!!!HashSign!!!","#")
              :gsub("!!!LBrace!!!",  "{")
              :gsub("!!!RBrace!!!",  "}")
  end
end

luamplib.everymplib    = setmetatable({ [""] = "" },{ __index = function(t) return t[""] end })
luamplib.everyendmplib = setmetatable({ [""] = "" },{ __index = function(t) return t[""] end })

local function process_mplibcode (data, instancename)
  texboxes.localid = 4096
```

This is needed for legacy behavior

```
  if luamplib.legacy_verbatimtex then
    luamplib.figid, tex_code_pre_mplib = 1, {}
  end

  local everymplib    = luamplib.everymplib[instancename]
  local everyendmplib = luamplib.everyendmplib[instancename]
  data = format("\n%s\n%s\n%s\n",everymplib, data, everyendmplib)
  :gsub("\r","\n")
```

These five lines are needed for `mplibverbatim` mode.

```
  if luamplib.verbatiminput then
    data = data:gsub("\\mpcolor%s+(.-%b{})","mplibcolor(\"%1\")")
    :gsub("\\mpdim%s+(%b{})", "mplibdimen(\"%1\")")
    :gsub("\\mpdim%s+(\\%a+)","mplibdimen(\"%1\")")
    :gsub(btex_etex, "btex %1 etex ")
    :gsub(verbatimtex_etex, "verbatimtex %1 etex;")
```

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.

```
  else
    data = data:gsub(btex_etex, function(str)
      return format("btex %s etex ", protect_expansion(str)) -- space
    end)
    :gsub(verbatimtex_etex, function(str)
      return format("verbatimtex %s etex;", protect_expansion(str)) -- semicolon
    end)
    :gsub("\".-\"", protect_expansion)
    :gsub("\\%%", "\0PerCent\0")
    :gsub("%%.-\n","\n")
    :gsub("%zPerCent%z", "\\%%")
    run_tex_code(format("\\mplibtmptoks\\expandafter{\\expanded{%s}}",data))
    data = texgettoks"mplibtmptoks"
```

Next line to address issue #55

```
    :gsub("##", "#")
```

```
    :gsub("\".-\"", unprotect_expansion)
    :gsub(btex_etex, function(str)
      return format("btex %s etex", unprotect_expansion(str))
    end)
    :gsub(verbatimtex_etex, function(str)
      return format("verbatimtex %s etex", unprotect_expansion(str))
    end)
  end

  process(data, instancename)
end
luamplib.process_mplibcode = process_mplibcode

```

For parsing prescript materials.

```
local further_split_keys = {
  mplibtexboxid = true,
  sh_color_a    = true,
  sh_color_b    = true,
}
local function script2table(s)
  local t = {}
  for _,i in ipairs(s:explode("\13+")) do
    local k,v = i:match("(.-)=(.*)") -- v may contain = or empty.
    if k and v and k ~= "" and not t[k] then
      if further_split_keys[k] or further_split_keys[k:sub(1,10)] then
        t[k] = v:explode(":")
      else
        t[k] = v
      end
    end
  end
  return t
end

```

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

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

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

local figcontents = { post = { } }
local function put2output(a,...)
  figcontents[#figcontents+1] = type(a) == "string" and format(a,...) or a
end

local function pdf_startfigure(n,llx,lly,urx,ury)
  put2output("\\mplibstarttoPDF{%f}{%f}{%f}{%f}",llx,lly,urx,ury)
```

```
end

local function pdf_stopfigure()
  put2output("\\mplibstoptoPDF")
end

```

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

```
local function pdf_literalcode (fmt,...)
  put2output{-2, format(fmt,...)}
end

local function pdf_textfigure(font,size,text,width,height,depth)
  text = text:gsub(".",function(c)
    return format("\\hbox{\\char%i}",string.byte(c)) -- kerning happens in metapost : false
  end)
  put2output("\\mplibtextext{%s}{%f}{%s}{%s}{%s}",font,size,text,0,0)
end

local bend_tolerance = 131/65536

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

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

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

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

local function flushnormalpath(path,open)
  local pth, ith
  for i=1,#path do
    pth = path[i]
    if not ith then
      pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
    elseif curved(ith,pth) then
      pdf_literalcode("%f %f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,pth.y_coord)
```

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

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

local function start_pdf_code()
  if pdfmode then
    pdf_literalcode("q")
  else
    put2output"\\special{pdf:bcontent}"
```

```
  end
end
local function stop_pdf_code()
  if pdfmode then
    pdf_literalcode("Q")
  else
    put2output"\\special{pdf:econtent}"
  end
end

```

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

```
local function put_tex_boxes (object,prescript)
  local box = prescript.mplibtexboxid
  local n,tw,th = box[1],tonumber(box[2]),tonumber(box[3])
  if n and tw and th then
    local op = object.path
    local first, second, fourth = op[1], op[2], op[4]
    local tx, ty = first.x_coord, first.y_coord
    local sx, rx, ry, sy = 1, 0, 0, 1
    if tw ~= 0 then
      sx = (second.x_coord - tx)/tw
      rx = (second.y_coord - ty)/tw
      if sx == 0 then sx = 0.00001 end
    end
    if th ~= 0 then
      sy = (fourth.y_coord - ty)/th
      ry = (fourth.x_coord - tx)/th
      if sy == 0 then sy = 0.00001 end
    end
    start_pdf_code()
    pdf_literalcode("%f %f %f %f %f %f cm",sx,rx,ry,sy,tx,ty)
    put2output("\\mplibputtextbox{%i}",n)
    stop_pdf_code()
  end
end

```

Colors

```
local prev_override_color
local function do_preobj_CR(object,prescript)
  local override = prescript and prescript.MPlibOverrideColor
  if override then
    if pdfmode then
      pdf_literalcode(override)
      override = nil
    else
      put2output("\\special{%s}",override)
      prev_override_color = override
    end
  else
    local cs = object.color
    if cs and #cs > 0 then
      pdf_literalcode(luamplib.colorconverter(cs))
```

```
      prev_override_color = nil
    elseif not pdfmode then
      override = prev_override_color
      if override then
        put2output("\\special{%s}",override)
      end
    end
  end
  return override
end
```

For transparency and shading

```
local pdfmanagement = is_defined'pdfmanagement_add:nnn'
local pdfobjs, pdfetcs = {}, {}
pdfetcs.pgfextgs = "pgf@sys@addpdfresource@extgs@plain"

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

local function update_pdfobjs (os)
  local on = pdfobjs[os]
  if on then
    return on,false
  end
  if pdfmode then
    on = pdf.immediateobj(os)
  else
    on = pdfetcs.cnt or 1
    texsprint(format("\\special{pdf:obj @mplibpdfobj%s %s}",on,os))
    pdfetcs.cnt = on + 1
  end
  pdfobjs[os] = on
  return on,true
end
```

Transparency

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

local function opacity_initialize ()
  pdfetcs.opacity_res = {}
  if pdfmode and luatexbase.callbacktypes.finish_pdffile then -- ltluatex
    local extgstate_obj = pdf.reserveobj()
    pdfetcs.setpageres(format("%s/ExtGState %i 0 R",pdfetcs.getpageres() or "",extgstate_obj))
```

```
    luatexbase.add_to_callback("finish_pdffile", function()
      pdf.immediateobj(extgstate_obj, format("<<%s>>",tableconcat(pdfetcs.opacity_res)))
    end, "luamplib.opacity.finish_pdffile")
  end
end

local function update_tr_res(mode,opaq)
  if pdfetcs.pgfloaded == nil then
    pdfetcs.pgfloaded = is_defined(pdfetcs.pgfextgs)
    if not pdfmanagement and not pdfetcs.pgfloaded and not is_defined"TRP@list" then
      opacity_initialize()
    end
  end
  local os = format("<</BM /%s/ca %.3f/CA %.3f/AIS false>>",mode,opaq,opaq)
  local on, new = update_pdfobjs(os)
  if new then
    if pdfmode then
      if pdfmanagement then
        texsprint(ccexplat,{
          [[\pdfmanagement_add:nnn{Page/Resources/ExtGState}]],
          format("{MPlibTr%s}{%s 0 R}", on, on),
        })
      else
        local tr = format("/MPlibTr%s %s 0 R",on,on)
        if pdfetcs.pgfloaded then
          texsprint(format("\\csname %s\\endcsname{%s}", pdfetcs.pgfextgs,tr))
        elseif is_defined"TRP@list" then
          texsprint(catat11,{
            [[\if@filesw\immediate\write\@auxout{]],
            [[\string\g@addto@macro\string\TRP@list{]],
            tr,
            [[}}\fi]],
          })
          if not get_macro"TRP@list":find(tr) then
            texsprint(catat11,[[\global\TRP@reruntrue]])
          end
        else
          if luatexbase.callbacktypes.finish_pdffile then
            pdfetcs.opacity_res[#pdfetcs.opacity_res+1] = tr
          else
            local tpr, n = pdfetcs.getpageres() or "", 0
            tpr, n = tpr:gsub("/ExtGState<<", "%1"..tr)
            if n == 0 then
              tpr = format("%s/ExtGState<<%s>>", tpr, tr)
            end
            pdfetcs.setpageres(tpr)
          end
        end
      end
    else
      if pdfmanagement then
        texsprint(ccexplat,{
          [[\pdfmanagement_add:nnn{Page/Resources/ExtGState}]],
          format("{MPlibTr%s}{@mplibpdfobj%s}", on, on),
```

```
        })
      else
        local tr = format("/MPlibTr%s @mplibpdfobj%s",on,on)
        if pdfetcs.pgfloaded then
          texsprint(format("\\csname %s\\endcsname{%s}", pdfetcs.pgfextgs,tr))
        else
          texsprint(format("\\special{pdf:put @MPlibTr<<%s>>}",tr))
          texsprint"\\special{pdf:put @resources<</ExtGState @MPlibTr>>}"
        end
      end
    end
  end
  return on
end

local function do_preobj_TR(prescript)
  local opaq = prescript and prescript.tr_transparency
  local tron_no
  if opaq then
    local mode = prescript.tr_alternative or 1
    mode = transparancy_modes[tonumber(mode)]
    tron_no = update_tr_res(mode, opaq)
    start_pdf_code()
    pdf_literalcode("/MPlibTr%i gs",tron_no)
  end
  return tron_no
end

```

Shading with metafun format.

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

local function sh_pdfpageresources(shtype,domain,colorspace,ca,cb,coordinates,steps,fractions)
  if not pdfmanagement and not pdfetcs.shading_res then
    shading_initialize()
  end
  local fun2fmt,os = "<</FunctionType 2/Domain [%s]/C0 [%s]/C1 [%s]/N 1>>"
  if steps > 1 then
    local list,bounds,encode = { },{ },{ }
    for i=1,steps do
      if i < steps then
        bounds[i] = fractions[i] or 1
      end
      encode[2*i-1] = 0
      encode[2*i]   = 1
      os = fun2fmt:format(domain,tableconcat(ca[i],' '),tableconcat(cb[i],' '))
```

```
    list[i] = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s",update_pdfobjs(os))
  end
  os = tableconcat {
    "<</FunctionType 3",
    format("/Bounds [%s]",    tableconcat(bounds," ")),
    format("/Encode [%s]",    tableconcat(encode," ")),
    format("/Functions [%s]", tableconcat(list,  " ")),
    format("/Domain [%s]>>",  domain),
  }
else
  os = fun2fmt:format(domain,tableconcat(ca[1]," "),tableconcat(cb[1]," "))
end
local objref = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s",update_pdfobjs(os))
os = tableconcat {
  format("<</ShadingType %i", shtype),
  format("/ColorSpace %s",    colorspace),
  format("/Function %s",      objref),
  format("/Coords [%s]",      coordinates),
  "/Extend [true true]/AntiAlias true>>",
}
local on, new = update_pdfobjs(os)
if pdfmode then
  if new then
    if pdfmanagement then
      texsprint(ccexplat,{
        [[\pdfmanagement_add:nnn{Page/Resources/Shading}]],
        format("{MPlibSh%s}{%s 0 R}", on, on),
      })
    else
      local res = format("/MPlibSh%s %s 0 R", on, on)
      if luatexbase.callbacktypes.finish_pdffile then
        pdfetcs.shading_res[#pdfetcs.shading_res+1] = res
      else
        local pageres = pdfetcs.getpageres() or ""
        if not pageres:find("/Shading<<.*>>") then
          pageres = pageres.."/Shading<<>>"
        end
        pageres = pageres:gsub("/Shading<<","%1"..res)
        pdfetcs.setpageres(pageres)
      end
    end
  end
else
  if pdfmanagement then
    if new then
      texsprint(ccexplat,{
        [[\pdfmanagement_add:nnn{Page/Resources/Shading}]],
        format("{MPlibSh%s}{@mplibpdfobj%s}", on, on),
      })
    end
  else
    if new then
      texsprint{
        "\\special{pdf:put @MPlibSh",
```

```
          format("<</MPlibSh%s @mplibpdfobj%s>>}",on, on),
        }
      end
      texsprint"\\special{pdf:put @resources<</Shading @MPlibSh>>}"
    end
  end
  return on
end

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

pdfetcs.clrspcs = { }
local function do_preobj_SH(object,prescript)
  local shade_no
  local sh_type = prescript and prescript.sh_type
  if sh_type then
    local domain  = prescript.sh_domain or "0 1"
    local centera = prescript.sh_center_a or "0 0"; centera = centera:explode()
    local centerb = prescript.sh_center_b or "0 0"; centerb = centerb:explode()
    local transform = prescript.sh_transform == "yes"
    local sx,sy,sr,dx,dy = 1,1,1,0,0
    if transform then
      local first = prescript.sh_first or "0 0"; first = first:explode()
      local setx = prescript.sh_set_x or "0 0";  setx = setx:explode()
      local sety = prescript.sh_set_y or "0 0";  sety = sety:explode()
      local x,y = tonumber(setx[1]) or 0, tonumber(sety[1]) or 0
      if x ~= 0 and y ~= 0 then
        local path = object.path
        local path1x = path[1].x_coord
        local path1y = path[1].y_coord
        local path2x = path[x].x_coord
        local path2y = path[y].y_coord
        local dxa = path2x - path1x
        local dya = path2y - path1y
        local dxb = setx[2] - first[1]
        local dyb = sety[2] - first[2]
        if dxa ~= 0 and dya ~= 0 and dxb ~= 0 and dyb ~= 0 then
          sx = dxa / dxb ; if sx < 0 then sx = - sx end
          sy = dya / dyb ; if sy < 0 then sy = - sy end
          sr = math.sqrt(sx^2 + sy^2)
          dx = path1x - sx*first[1]
          dy = path1y - sy*first[2]
        end
      end
```

```
  end
  local ca, cb, colorspace, steps, fractions
  ca = { prescript.sh_color_a_1 or prescript.sh_color_a or {0} }
  cb = { prescript.sh_color_b_1 or prescript.sh_color_b or {1} }
  steps = tonumber(prescript.sh_step) or 1
  if steps > 1 then
    fractions = { prescript.sh_fraction_1 or 0 }
    for i=2,steps do
      fractions[i] = prescript[format("sh_fraction_%i",i)] or (i/steps)
      ca[i] = prescript[format("sh_color_a_%i",i)] or {0}
      cb[i] = prescript[format("sh_color_b_%i",i)] or {1}
    end
  end
  if prescript.mplib_spotcolor then
    ca, cb = { }, { }
    local names, pos, objref = { }, -1, ""
    local script = object.prescript:explode"\13+"
    for i=#script,1,-1 do
      if script[i]:find"mplib_spotcolor" then
        local name, value
        objref, name = script[i]:match"=(.-):(.+)"
        value = script[i+1]:match"=(.+)"
        if not names[name] then
          pos = pos+1
          names[name] = pos
          names[#names+1] = name
        end
        local t = { }
        for j=1,names[name] do t[#t+1] = 0 end
        t[#t+1] = value
        table.insert(#ca == #cb and ca or cb, t)
      end
    end
    for _,t in ipairs{ca,cb} do
      for _,tt in ipairs(t) do
        for i=1,#names-#tt do tt[#tt+1] = 0 end
      end
    end
    if #names == 1 then
      colorspace = objref
    else
      local name = tableconcat(names,"-")
      local obj = pdfetcs.clrspcs[name]
      if obj then
        colorspace = obj
      else
        run_tex_code({
          [[\color_model_new:nnn]],
          format("{mplibcolorspace_%s}", name),
          format("{DeviceN}{names={%s}}", tableconcat(names,",")),
          [[\edef\mplib_@tempa{\pdf_object_ref_last:}]],
        }, ccexplat)
        colorspace = get_macro'mplib_@tempa'
        pdfetcs.clrspcs[name] = colorspace
```

```
      end
    end
  else
    local model = 0
    for _,t in ipairs{ca,cb} do
      for _,tt in ipairs(t) do
        model = model > #tt and model or #tt
      end
    end
    for _,t in ipairs{ca,cb} do
      for _,tt in ipairs(t) do
        if #tt < model then
          color_normalize(model == 4 and {1,1,1,1} or {1,1,1},tt)
        end
      end
    end
    colorspace = model == 4 and "/DeviceCMYK"
              or model == 3 and "/DeviceRGB"
              or model == 1 and "/DeviceGray"
              or err"unknown color model"
  end
  if sh_type == "linear" then
    local coordinates = format("%f %f %f %f",
      dx + sx*centera[1], dy + sy*centera[2],
      dx + sx*centerb[1], dy + sy*centerb[2])
    shade_no = sh_pdfpageresources(2,domain,colorspace,ca,cb,coordinates,steps,fractions)
  elseif sh_type == "circular" then
    local factor = prescript.sh_factor or 1
    local radiusa = factor * prescript.sh_radius_a
    local radiusb = factor * prescript.sh_radius_b
    local coordinates = format("%f %f %f %f %f %f",
      dx + sx*centera[1], dy + sy*centera[2], sr*radiusa,
      dx + sx*centerb[1], dy + sy*centerb[2], sr*radiusb)
    shade_no = sh_pdfpageresources(3,domain,colorspace,ca,cb,coordinates,steps,fractions)
  else
    err"unknown shading type"
  end
  pdf_literalcode("q /Pattern cs")
 end
 return shade_no
end
```

Finally, flush figures by inserting PDF literals.

```
local function flush(result,flusher)
 if result then
   local figures = result.fig
   if figures then
     for f=1, #figures do
       info("flushing figure %s",f)
       local figure = figures[f]
       local objects = getobjects(result,figure,f)
       local fignum = tonumber(figure:filename():match("([%d]+)$") or figure:charcode() or 0)
       local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
       local bbox = figure:boundingbox()
```

```
        local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
        if urx < llx then
```

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()
```

```
        else
```

For legacy behavior, insert 'pre-fig' TEX code here.

```
          if tex_code_pre_mplib[f] then
            put2output(tex_code_pre_mplib[f])
          end
          pdf_startfigure(fignum,llx,lly,urx,ury)
          start_pdf_code()
          if objects then
            local savedpath = nil
            local savedhtap = nil
            for o=1,#objects do
              local object        = objects[o]
              local objecttype    = object.type
```

The following 6 lines are part of btex...etex patch. Again, colors are processed at this stage.

```
              local prescript     = object.prescript
              prescript = prescript and script2table(prescript) -- prescript is now a table
              local cr_over = do_preobj_CR(object,prescript) -- color
              local tr_opaq = do_preobj_TR(prescript) -- opacity
              if prescript and prescript.mplibtexboxid then
                put_tex_boxes(object,prescript)
              elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then --skip
              elseif objecttype == "start_clip" then
                local evenodd = not object.istext and object.postscript == "evenodd"
                start_pdf_code()
                flushnormalpath(object.path,false)
                pdf_literalcode(evenodd and "W* n" or "W n")
              elseif objecttype == "stop_clip" then
                stop_pdf_code()
                miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
              elseif objecttype == "special" then
```

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

```
                if prescript and prescript.postmplibverbtex then
                  figcontents.post[#figcontents.post+1] = prescript.postmplibverbtex
                end
              elseif objecttype == "text" then
                local ot = object.transform -- 3,4,5,6,1,2
                start_pdf_code()
                pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
                pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
                stop_pdf_code()
              else
```

```
            local evenodd, collect, both = false, false, false
            local postscript = object.postscript
            if not object.istext then
              if postscript == "evenodd" then
                evenodd = true
              elseif postscript == "collect" then
                collect = true
              elseif postscript == "both" then
                both = true
              elseif postscript == "eoboth" then
                evenodd = true
                both    = true
              end
            end
            if collect then
              if not savedpath then
                savedpath = { object.path or false }
                savedhtap = { object.htap or false }
              else
                savedpath[#savedpath+1] = object.path or false
                savedhtap[#savedhtap+1] = object.htap or false
              end
            else
```

Removed from ConTeXt general: color stuff. Added instead : shading stuff

```
              local shade_no = do_preobj_SH(object,prescript) -- shading
              local ml = object.miterlimit
              if ml and ml ~= miterlimit then
                miterlimit = ml
                pdf_literalcode("%f M",ml)
              end
              local lj = object.linejoin
              if lj and lj ~= linejoin then
                linejoin = lj
                pdf_literalcode("%i j",lj)
              end
              local lc = object.linecap
              if lc and lc ~= linecap then
                linecap = lc
                pdf_literalcode("%i J",lc)
              end
              local dl = object.dash
              if dl then
                local d = format("[%s] %f d",tableconcat(dl.dashes or {}," "),dl.offset)
                if d ~= dashed then
                  dashed = d
                  pdf_literalcode(dashed)
                end
              elseif dashed then
                pdf_literalcode("[] 0 d")
                dashed = false
              end
              local path = object.path
              local transformed, penwidth = false, 1
              local open = path and path[1].left_type and path[#path].right_type
```

```
1553                local pen = object.pen
1554                if pen then
1555                  if pen.type == 'elliptical' then
1556                    transformed, penwidth = pen_characteristics(object) -- boolean, value
1557                    pdf_literalcode("%f w",penwidth)
1558                    if objecttype == 'fill' then
1559                      objecttype = 'both'
1560                    end
1561                  else -- calculated by mplib itself
1562                    objecttype = 'fill'
1563                  end
1564                end
1565                if transformed then
1566                  start_pdf_code()
1567                end
1568                if path then
1569                  if savedpath then
1570                    for i=1,#savedpath do
1571                      local path = savedpath[i]
1572                      if transformed then
1573                        flushconcatpath(path,open)
1574                      else
1575                        flushnormalpath(path,open)
1576                      end
1577                    end
1578                    savedpath = nil
1579                  end
1580                  if transformed then
1581                    flushconcatpath(path,open)
1582                  else
1583                    flushnormalpath(path,open)
1584                  end
```

Shading seems to conflict with these ops

```
1585                  if not shade_no then -- conflict with shading
1586                    if objecttype == "fill" then
1587                      pdf_literalcode(evenodd and "h f*" or "h f")
1588                    elseif objecttype == "outline" then
1589                      if both then
1590                        pdf_literalcode(evenodd and "h B*" or "h B")
1591                      else
1592                        pdf_literalcode(open and "S" or "h S")
1593                      end
1594                    elseif objecttype == "both" then
1595                      pdf_literalcode(evenodd and "h B*" or "h B")
1596                    end
1597                  end
1598                end
1599                if transformed then
1600                  stop_pdf_code()
1601                end
1602                local path = object.htap
1603                if path then
1604                  if transformed then
1605                    start_pdf_code()
```

```
                end
                if savedhtap then
                  for i=1,#savedhtap do
                    local path = savedhtap[i]
                    if transformed then
                      flushconcatpath(path,open)
                    else
                      flushnormalpath(path,open)
                    end
                  end
                  savedhtap = nil
                  evenodd   = true
                end
                if transformed then
                  flushconcatpath(path,open)
                else
                  flushnormalpath(path,open)
                end
                if objecttype == "fill" then
                  pdf_literalcode(evenodd and "h f*" or "h f")
                elseif objecttype == "outline" then
                  pdf_literalcode(open and "S" or "h S")
                elseif objecttype == "both" then
                  pdf_literalcode(evenodd and "h B*" or "h B")
                end
                if transformed then
                  stop_pdf_code()
                end
              end
```

Added to ConTeXt general: post-object color and shading stuff.

```
                if shade_no then -- shading
                  pdf_literalcode("W n /MPlibSh%s sh Q",shade_no)
                end
              end
            end
            if tr_opaq then -- opacity
              stop_pdf_code()
            end
            if cr_over then -- color
              put2output"\\special{pdf:ec}"
            end
          end
        end
        stop_pdf_code()
        pdf_stopfigure()
```

output collected materials to PDF, plus legacy `verbatimtex` code.

```
        for _,v in ipairs(figcontents) do
          if type(v) == "table" then
            texsprint"\\mplibtoPDF{"; texsprint(v[1], v[2]); texsprint"}"
          else
            texsprint(v)
          end
        end
```

```
          if #figcontents.post > 0 then texsprint(figcontents.post) end
          figcontents = { post = { } }
        end
      end
    end
  end
end
luamplib.flush = flush

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

2.2 TEX package

First we need to load some packages.

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname selectfont\endcsname\relax
  \input ltluatex
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luamplib}
    [2024/05/01 v2.29.0 mplib package for LuaTeX]
  \ifx\newluafunction\@undefined
  \input ltluatex
  \fi
\fi
```

Loading of lua code.

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

legacy commands. Seems we don't need it, but no harm.

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

Set the format for metapost.

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

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

```
\ifnum\pdfoutput>0
  \let\mplibtoPDF\pdfliteral
\else
  \def\mplibtoPDF#1{\special{pdf:literal direct #1}}
  \ifcsname PackageInfo\endcsname
    \PackageInfo{luamplib}{only dvipdfmx is supported currently}
  \else
    \immediate\write-1{luamplib Info: only dvipdfmx is supported currently}
  \fi
\fi
```

To make mplibcode typeset always in horizontal mode.

```
\def\mplibforcehmode{\let\prependtomplibbox\leavevmode}
\def\mplibnoforcehmode{\let\prependtomplibbox\relax}
\mplibnoforcehmode
```

Catcode. We want to allow comment sign in mplibcode.

```
\def\mplibsetupcatcodes{%
  %catcode`\{=12 %catcode`\}=12
  \catcode`\#=12 \catcode`\^=12 \catcode`\~=12 \catcode`\_=12
  \catcode`\&=12 \catcode`\$=12 \catcode`\%=12 \catcode`\^^M=12
}
```

Make btex...etex box zero-metric.

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

simple way to use mplib: `\mpfig draw fullcircle scaled 10; \endmpfig`

```
\def\mpfiginstancename{@mpfig}
\protected\def\mpfig{%
  \begingroup
  \futurelet\nexttok\mplibmpfigbranch
}
\def\mplibmpfigbranch{%
  \ifx *\nexttok
    \expandafter\mplibprempfig
  \else
    \expandafter\mplibmainmpfig
  \fi
}
\def\mplibmainmpfig{%
  \begingroup
  \mplibsetupcatcodes
  \mplibdomainmpfig
}
\long\def\mplibdomainmpfig#1\endmpfig{%
  \endgroup
  \directlua{
    local legacy = luamplib.legacy_verbatimtex
    local everympfig = luamplib.everymplib["\mpfiginstancename"] or ""
    local everyendmpfig = luamplib.everyendmplib["\mpfiginstancename"] or ""
    luamplib.legacy_verbatimtex = false
    luamplib.everymplib["\mpfiginstancename"] = ""
    luamplib.everyendmplib["\mpfiginstancename"] = ""
    luamplib.process_mplibcode(
    "beginfig(0) "..everympfig.." "..[===[\unexpanded{#1}]===].." "..everyendmpfig.." endfig;",
    "\mpfiginstancename")
```

```
    luamplib.legacy_verbatimtex = legacy
    luamplib.everymplib["\mpfiginstancename"] = everympfig
    luamplib.everyendmplib["\mpfiginstancename"] = everyendmpfig
  }%
  \endgroup
}
\def\mplibprempfig#1{%
  \begingroup
  \mplibsetupcatcodes
  \mplibdoprempfig
}
\long\def\mplibdoprempfig#1\endmpfig{%
  \endgroup
  \directlua{
    local legacy = luamplib.legacy_verbatimtex
    local everympfig = luamplib.everymplib["\mpfiginstancename"]
    local everyendmpfig = luamplib.everyendmplib["\mpfiginstancename"]
    luamplib.legacy_verbatimtex = false
    luamplib.everymplib["\mpfiginstancename"] = ""
    luamplib.everyendmplib["\mpfiginstancename"] = ""
    luamplib.process_mplibcode([===[\unexpanded{#1}]===],"\mpfiginstancename")
    luamplib.legacy_verbatimtex = legacy
    luamplib.everymplib["\mpfiginstancename"] = everympfig
    luamplib.everyendmplib["\mpfiginstancename"] = everyendmpfig
  }%
  \endgroup
}
\protected\def\endmpfig{endmpfig}
```

The Plain-specific stuff.

```
\unless\ifcsname ver@luamplib.sty\endcsname
  \def\mplibcodegetinstancename[#1]{\gdef\currentmpinstancename{#1}\mplibcodeindeed}
  \protected\def\mplibcode{%
    \begingroup
    \futurelet\nexttok\mplibcodebranch
  }
  \def\mplibcodebranch{%
    \ifx [\nexttok
      \expandafter\mplibcodegetinstancename
    \else
      \global\let\currentmpinstancename\empty
      \expandafter\mplibcodeindeed
    \fi
  }
  \def\mplibcodeindeed{%
    \begingroup
    \mplibsetupcatcodes
    \mplibdocode
  }
  \long\def\mplibdocode#1\endmplibcode{%
    \endgroup
    \directlua{luamplib.process_mplibcode([===[\unexpanded{#1}]===],"\currentmpinstancename")}%
    \endgroup
  }
  \protected\def\endmplibcode{endmplibcode}
```

```
\else
```

The LaTeX-specific part: a new environment.

```
  \newenvironment{mplibcode}[1][]{%
    \global\def\currentmpinstancename{#1}%
    \mplibtmptoks{}\ltxdomplibcode
  }{}
  \def\ltxdomplibcode{%
    \begingroup
    \mplibsetupcatcodes
    \ltxdomplibcodeindeed
  }
  \def\mplib@mplibcode{mplibcode}
  \long\def\ltxdomplibcodeindeed#1\end#2{%
    \endgroup
    \mplibtmptoks\expandafter{\the\mplibtmptoks#1}%
    \def\mplibtemp@a{#2}%
    \ifx\mplib@mplibcode\mplibtemp@a
      \directlua{luamplib.process_mplibcode([===[\the\mplibtmptoks]===],"\currentmpinstancename")}%
      \end{mplibcode}%
    \else
      \mplibtmptoks\expandafter{\the\mplibtmptoks\end{#2}}%
      \expandafter\ltxdomplibcode
    \fi
  }
\fi
```

User settings.

```
\def\mplibshowlog#1{\directlua{
    local s = string.lower("#1")
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.showlog = true
    else
      luamplib.showlog = false
    end
}}
\def\mpliblegacybehavior#1{\directlua{
    local s = string.lower("#1")
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.legacy_verbatimtex = true
    else
      luamplib.legacy_verbatimtex = false
    end
}}
\def\mplibverbatim#1{\directlua{
    local s = string.lower("#1")
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.verbatiminput = true
    else
      luamplib.verbatiminput = false
    end
}}
\newtoks\mplibtmptoks
```

\everymplib & \everyendmplib: macros resetting luamplib.every(end)mplib tables

```
\ifcsname ver@luamplib.sty\endcsname
  \protected\def\everymplib{%
    \begingroup
    \mplibsetupcatcodes
    \mplibdoeverymplib
  }
  \protected\def\everyendmplib{%
    \begingroup
    \mplibsetupcatcodes
    \mplibdoeveryendmplib
  }
  \newcommand\mplibdoeverymplib[2][]{%
    \endgroup
    \directlua{
      luamplib.everymplib["#1"] = [===[\unexpanded{#2}]===]
    }%
  }
  \newcommand\mplibdoeveryendmplib[2][]{%
    \endgroup
    \directlua{
      luamplib.everyendmplib["#1"] = [===[\unexpanded{#2}]===]
    }%
  }
\else
  \def\mplibgetinstancename[#1]{\def\currentmpinstancename{#1}}
  \protected\def\everymplib#1#{%
    \ifx\empty#1\empty \mplibgetinstancename[]\else \mplibgetinstancename#1\fi
    \begingroup
    \mplibsetupcatcodes
    \mplibdoeverymplib
  }
  \long\def\mplibdoeverymplib#1{%
    \endgroup
    \directlua{
      luamplib.everymplib["\currentmpinstancename"] = [===[\unexpanded{#1}]===]
    }%
  }
  \protected\def\everyendmplib#1#{%
    \ifx\empty#1\empty \mplibgetinstancename[]\else \mplibgetinstancename#1\fi
    \begingroup
    \mplibsetupcatcodes
    \mplibdoeveryendmplib
  }
  \long\def\mplibdoeveryendmplib#1{%
    \endgroup
    \directlua{
      luamplib.everyendmplib["\currentmpinstancename"] = [===[\unexpanded{#1}]===]
    }%
  }
\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.

```
\def\mpdim#1{ runscript("luamplibdimen{#1}") }
\def\mpcolor#1#{\domplibcolor{#1}}
\def\domplibcolor#1#2{ runscript("luamplibcolor{#1{#2}}") }
```

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

```
\def\mplibnumbersystem#1{\directlua{
  local t = "#1"
  if t == "binary" then t = "decimal" end
  luamplib.numbersystem = t
}}
```

Settings for `.mp` cache files.

```
\def\mplibmakenocache#1{\mplibdomakenocache #1,*,}
\def\mplibdomakenocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdomakenocache
  \else
    \ifx*#1\else
      \directlua{luamplib.noneedtoreplace["#1.mp"]=true}%
      \expandafter\expandafter\expandafter\mplibdomakenocache
    \fi
  \fi
}
\def\mplibcancelnocache#1{\mplibdocancelnocache #1,*,}
\def\mplibdocancelnocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdocancelnocache
  \else
    \ifx*#1\else
      \directlua{luamplib.noneedtoreplace["#1.mp"]=false}%
      \expandafter\expandafter\expandafter\mplibdocancelnocache
    \fi
  \fi
}
\def\mplibcachedir#1{\directlua{luamplib.getcachedir("\unexpanded{#1}")}}
```

More user settings.

```
\def\mplibtextextlabel#1{\directlua{
    local s = string.lower("#1")
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.textextlabel = true
    else
      luamplib.textextlabel = false
    end
}}
\def\mplibcodeinherit#1{\directlua{
    local s = string.lower("#1")
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.codeinherit = true
    else
      luamplib.codeinherit = false
    end
}}
\def\mplibglobaltextext#1{\directlua{
    local s = string.lower("#1")
```

```
    if s == "enable" or s == "true" or s == "yes" then
      luamplib.globaltextext = true
    else
      luamplib.globaltextext = false
    end
}}
```

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

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

We encapsulate the litterals.

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

Text items have a special handler.

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

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

Input `luamplib.cfg` when it exists.

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

That's all folks!

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

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

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 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.

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 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.

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 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.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) 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,

   (b) 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,

   (c) 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.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided 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.

6. 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.

7. 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.

8. 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 royalty-free redistribution of 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.

9. 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.

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 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.

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. 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

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 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.

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 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 WITH-
OUT ANY WARRANTY; without even the implied warranty of MER-
CHANTABILITY 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 cer-
tain 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:

```
Yoyodyne, 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.