

# The luamplib package

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

2024/06/13 v2.32.1

## Abstract

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

## 1 Documentation

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

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

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

Using this package is easy: in Plain, type your metapost code between the macros `\mplibcode` and `\endmpplibcode`, and in  $\LaTeX$  in the `mpplibcode` environment.

The code is from the `luatex-mpplib.lua` and `luatex-mpplib.tex` files from ConTeXt, they have been adapted to  $\LaTeX$  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 `mpplib`
- use of our own function for errors, warnings and informations
- possibility to use `btex ... etex` to typeset TeX code. `texttext()` is a more versatile macro equivalent to `TEX()` from `TEX.mp`. `TEX()` is also allowed and is a synonym of `texttext()`.

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

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

Some more changes and cautions are:

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

**\mpfig... \endmpfig** Since v2.29 we provide unexpandable T<sub>E</sub>X 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.<sup>1</sup>

**\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 T<sub>E</sub>X 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
```

---

<sup>1</sup>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 inspired by `gmp.sty` authored by Enrico Gregorio. Please refer the manual of `gmp` package for details.

```
\begin{mplibcode}
  draw origin--(.6\mpdim{\linewidth},0) withpen pencircle scaled 4
  dashed evenly scaled 4 withcolor \mpcolor{orange};
\end{mplibcode}
```

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

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

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

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

**\mplibtexttextlabel** Starting with v2.6, `\mplibtexttextlabel{enable}` enables string labels typeset via `texttext()` instead of `infont` operator. So, `label("my text",origin)` thereafter is exactly the same as `label(texttext("my text"),origin)`. N.B. In the background, `luamplib` redefines `infont` operator so that the right side argument (the font part) is totally ignored. Every string label therefore will be typeset with current  $\TeX$  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 `\mplibglobaltexttext`.
- When an instance names is set, respective `\currentmpinstancename` is set.

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

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

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

```

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

```

Generally speaking, it is recommended to turn `mplibglobaltexttext` always on, because it has the advantage of reusing metapost pictures among code chunks. 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 necessary, before returning their paths to Lua $\TeX$ '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, mplibrbgtexcolor** `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, `mplibrbgtexcolor` always returns rgb model expressions.

**mplibgraphicstext** For some amusement, `luamplib` provides its own metapost operator `mplibgraphicstext`, the effect of which is similar to that of `Con $\TeX$ t's` `graphicstext`. However syntax is somewhat different.

```
mplibgraphicstext "Funny"
  fakebold 2.3                % fontspec option
  drawcolor .7blue fillcolor "red!50" % color expressions
```

`fakebold`, `drawcolor` and `fillcolor` are optional; default values are 2, "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). From v2.30, `scale` option is deprecated and is now a synonym of `scaled`. All from `mplibgraphicstext` 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 `graphicstext`. N.B. Because `luamplib's` current implementation is quite different from the `Con $\TeX$ t's`, there are some limitations such that you can't apply shading (gradient colors) to the text (But see below). In DVI mode, `unicode-math` package is needed for math formula `graphicstext`, as we cannot embolden `type1` fonts in DVI mode.

**mplibglyph, mplibdrawglyph** From v2.30, we provide a new metapost operator `mplibglyph`, which returns a metapost picture containing outline paths of a glyph in `opentype`, `true-type` or `type1` fonts. When a `type1` font is specified, metapost primitive `glyph` will be called.

```
mplibglyph 50 of \fontid\font          % slot 50 of current font
mplibglyph "Q" of "TU/TeXGyrePagella(0)/m/n/10" % font csname
mplibglyph "Q" of "texgyrepagella-regular.otf" % raw filename
mplibglyph "Q" of "Times.ttc(2)"          % subfont number
mplibglyph "Q" of "SourceHanSansK-VF.otf[Regular]" % instance name
```

Both arguments before and after of "of" can be either a number or a string. Number arguments are regarded as a glyph slot (GID) and a font id number, respectively. String argument at the left side is regarded as a glyph name in the font or a unicode character. String argument at the right side is regarded as a  $\TeX$  font csname (without backslash) or the raw filename of a font. When it is a font filename, a number within parentheses after the filename denotes a subfont number (starting from zero) of a TTC font; a string within brackets denotes an instance name of a variable font.

The returned picture will be quite similar to the result of `glyph` primitive in its structure. So, `metapost`'s `draw` command will fill the inner path of the picture with background color. In contrast, `mplibdrawglyph` command fills the paths according to the Nonzero Winding Number Rule. As a result, for instance, the area surrounded by inner path of "O" will remain transparent.

**`mpliboutlinetext`** From v2.31, we provide a new `metapost` operator `mpliboutlinetext`, which mimicks `metafun`'s `outlinetext`. So the syntax is the same as `metafun`'s. See the `metafun` manual § 8.7 (`texdoc metafun`). A simple example:

```
draw mpliboutlinetext.b ("$\sqrt{2+\alpha}$")
  (withcolor \mpcolor{red!50})
  (withpen pencircle scaled .2 withcolor red)
  scaled 2 ;
```

After the process of `mpliboutlinetext`, `mpliboutlinepic[]` and `mpliboutlinenum` will be preserved as global variables; `mpliboutlinepic[1] ... mpliboutlinepic[mpliboutlinenum]` will be an array of images each of which containing a glyph or a rule. N.B. As Unicode grapheme cluster is not considered in the array, a unit that must be a single cluster might be separated apart.

**`\mppattern ... \endmppattern`, `withpattern`** `\mppattern{<name>} ... \endmppattern` defines a tiling pattern associated with the `<name>`. `MetaPost` operator `withpattern`, the syntax being `path withpattern string`, will return a `metapost` picture which fills the given path with a tiling pattern of the `<name>`.

```
\mppattern{mypatt}          % or \begin{mppattern}{mypatt}
[
  xstep = 10, ystep = 12,
  matrix = {0,1,-1,0},     % or "0 1 -1 0"
]
\mpfig                      % or any other TeX code,
  picture q;
  q := btex Q etex;
  fill bbox q withcolor .8[red,white];
  draw q withcolor .8red;
\endmpfig
\endmppattern              % or \end{mppattern}

\mpfig
  fill fullcircle scaled 100 withpostscript "collect";
  draw unitsquare shifted - center unitsquare scaled 45
  withpattern "mypatt"
  withpostscript "evenodd" ;
\endmpfig
```

The available options are:

Key	Value Type	Explanation
xstep	<i>number</i>	horizontal spacing between pattern cells
ystep	<i>number</i>	vertical spacing between pattern cells
xshift	<i>number</i>	horizontal shifting of pattern cells
yshift	<i>number</i>	vertical shifting of pattern cells
matrix	<i>table</i> or <i>string</i>	xx, xy, yx, yy values* or MP transformation code
bbox	<i>table</i> or <i>string</i>	llx, lly, urx, ury values*
resources	<i>string</i>	PDF resources if needed
colored	<i>boolean</i>	false for uncolored pattern. default: true

\* in string type, numbers are separated by spaces

For the sake of convenience, width and height values of tiling patterns will be written down into the log file. (depth is always zero.) Users can refer to them for option setting.

As for `matrix` option, metapost code such as `'rotated 30 slanted .2'` is allowed as well as string or table of four numbers. You can also set `xshift` and `yshift` values by using `'shifted'` operator. But when `xshift` or `yshift` option is explicitly given, they have precedence over the effect of `'shifted'` operator.

When you use special effects such as transparency in a pattern, `resources` option is needed: for instance, `resources="/ExtGState 1 0 R"`. However, as `luamplib` automatically includes the resources of the current page, this option is not needed in most cases.

Option `colored=false` will generate an uncolored pattern which shall have no color at all. Uncolored pattern will be painted later by the color of a metapost object. An example:

```

\begin{mppattern}{pattuncolored}
[
  colored = false,
  matrix = "rotated 30",
]
\tiny\TeX
\end{mppattern}

\begin{mplibcode}
beginfig(1)
picture tex; tex := mpliboutlinetext.p ("bfseries \TeX");
i:=0;
for item within tex:
  i:=i+1;
  if i < length tex:
    fill pathpart item scaled 10
      withpostsript "collect";
  else:
    draw pathpart item scaled 10
      withpattern "pattuncolored"
      withpen pencircle scaled 0.5
      withcolor 0.7 blue          % paints the pattern
    ;
  fi
endfor
endfig;
\end{mplibcode}

```

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

**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   = "2.32.1",
5   date      = "2024/06/13",
6   description = "Lua package to typeset Metapost with LuaTeX's MPLib.",
7 }
8
```

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

```

9 luamplib      = luamplib or { }
10 local luamplib = luamplib
11
12 local format, abs = string.format, math.abs
13
14 Use our own function for warn/info/err.
15 local function termorlog (target, text, kind)
16   if text then
17     local mod, write, append = "luamplib", texio.write_nl, texio.write
18     kind = kind
19     or target == "term" and "Warning (more info in the log)"
20     or target == "log" and "Info"
21     or target == "term and log" and "Warning"
22     or "Error"
23     target = kind == "Error" and "term and log" or target
24     local t = text:explode"\n+"
25     write(target, format("Module %s %s:", mod, kind))
26     if #t == 1 then
27       append(target, format(" %s", t[1]))
28     else
29       for _,line in ipairs(t) do
30         write(target, line)
31       end
32     write(target, format("(%s", mod))

```

```

32   end
33   append(target, format(" on input line %s", tex.inputlineno))
34   write(target, "")
35   if kind == "Error" then error() end
36 end
37 end
38
39 local function warn (...) -- beware '%' symbol
40   termorlog("term and log", select("#",...) > 1 and format(...) or ...)
41 end
42 local function info (...)
43   termorlog("log", select("#",...) > 1 and format(...) or ...)
44 end
45 local function err (...)
46   termorlog("error", select("#",...) > 1 and format(...) or ...)
47 end
48
49 luamplib.showlog = luamplib.showlog or false
50

```

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

```

51 local tableconcat = table.concat
52 local tableinsert = table.insert
53 local texsprintf = tex.sprintf
54 local texgettoks = tex.gettoks
55 local texgetbox = tex.getbox
56 local texruntoks = tex.runtoks

```

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

```

    local texscantoks = tex.scantoks

```

```

57
58 if not texruntoks then
59   err("Your LuaTeX version is too old. Please upgrade it to the latest")
60 end
61
62 local is_defined = token.is_defined
63 local get_macro = token.get_macro
64
65 local mplib = require ('mplib')
66 local kpse = require ('kpse')
67 local lfs = require ('lfs')
68
69 local lfsattributes = lfs.attributes
70 local lfsisdir = lfs.isdir
71 local lfsmkdir = lfs.mkdir
72 local lfstouch = lfs.touch
73 local iioopen = io.open
74

```

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

```

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

```

```

78 end
79
80 local is_writable = file.is_writable or function(name)
81   if lfs.isdir(name) then
82     name = name .. "/_luamplib_temp_file_"
83     local fh = io.open(name,"w")
84     if fh then
85       fh:close(); os.remove(name)
86       return true
87     end
88   end
89 end
90 local mk_full_path = lfs.mkdirp or lfs.mkdir or function(path)
91   local full = ""
92   for sub in path:gmatch("(/*[^\w/]+)") do
93     full = full .. sub
94     lfs.mkdir(full)
95   end
96 end
97

```

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.

```

98 local luamplibtime = kpse.find_file("luamplib.lua")
99 luamplibtime = luamplibtime and lfs.attributes(luamplibtime,"modification")
100
101 local currenttime = os.time()
102
103 local outputdir, cachedir
104 if lfstouch then
105   for i,v in ipairs{'TEXMFVAR', 'TEXMF_OUTPUT_DIRECTORY', '.', 'TEXMFOUTPUT'} do
106     local var = i == 3 and v or kpse.var_value(v)
107     if var and var ~= "" then
108       for _,vv in next, var:explode(os.type == "unix" and ":" or ";") do
109         local dir = format("%s/%s",vv,"luamplib_cache")
110         if not lfs.isdir(dir) then
111           mk_full_path(dir)
112         end
113         if is_writable(dir) then
114           outputdir = dir
115           break
116         end
117       end
118     if outputdir then break end
119   end
120 end
121 end
122 outputdir = outputdir or '.'
123 function luamplib.getcachedir(dir)
124   dir = dir:gsub("##", "#")
125   dir = dir:gsub("^~",
126     os.type == "windows" and os.getenv("UserProfile") or os.getenv("HOME"))
127   if lfstouch and dir then

```

```

128   if lfsisdir(dir) then
129       if is_writable(dir) then
130           cachedir = dir
131       else
132           warn("Directory '%s' is not writable!", dir)
133       end
134   else
135       warn("Directory '%s' does not exist!", dir)
136   end
137 end
138 end
139

```

Some basic MetaPost files not necessary to make cache files.

```

140 local noneedtoreplace = {
141   ["boxes.mp"] = true, -- ["format.mp"] = true,
142   ["graph.mp"] = true, ["marith.mp"] = true, ["mfplain.mp"] = true,
143   ["mpost.mp"] = true, ["plain.mp"] = true, ["rboxes.mp"] = true,
144   ["sarith.mp"] = true, ["string.mp"] = true, -- ["TEX.mp"] = true,
145   ["metafun.mp"] = true, ["metafun.mpiv"] = true, ["mp-abck.mpiv"] = true,
146   ["mp-apos.mpiv"] = true, ["mp-asnc.mpiv"] = true, ["mp-bare.mpiv"] = true,
147   ["mp-base.mpiv"] = true, ["mp-blob.mpiv"] = true, ["mp-butt.mpiv"] = true,
148   ["mp-char.mpiv"] = true, ["mp-chem.mpiv"] = true, ["mp-core.mpiv"] = true,
149   ["mp-crop.mpiv"] = true, ["mp-figs.mpiv"] = true, ["mp-form.mpiv"] = true,
150   ["mp-func.mpiv"] = true, ["mp-grap.mpiv"] = true, ["mp-grid.mpiv"] = true,
151   ["mp-grph.mpiv"] = true, ["mp-idea.mpiv"] = true, ["mp-luas.mpiv"] = true,
152   ["mp-mlib.mpiv"] = true, ["mp-node.mpiv"] = true, ["mp-page.mpiv"] = true,
153   ["mp-shap.mpiv"] = true, ["mp-step.mpiv"] = true, ["mp-text.mpiv"] = true,
154   ["mp-tool.mpiv"] = true, ["mp-cont.mpiv"] = true,
155 }
156 luamplib.noneedtoreplace = noneedtoreplace
157

```

format.mp is much complicated, so specially treated.

```

158 local function replaceformatmp(file,newfile,ofmodify)
159   local fh = ioopen(file,"r")
160   if not fh then return file end
161   local data = fh:read("*all"); fh:close()
162   fh = ioopen(newfile,"w")
163   if not fh then return file end
164   fh:write(
165     "let normalinfont = infont;\n",
166     "primarydef str infont name = rawtexttext(str) enddef;\n",
167     data,
168     "vardef Fmant_(expr x) = rawtexttext(decimal abs x) enddef;\n",
169     "vardef Fexp_(expr x) = rawtexttext(\"${\"&decimal x\"}$\") enddef;\n",
170     "let infont = normalinfont;\n"
171   ); fh:close()
172   ifstouch(newfile,currenttime,ofmodify)
173   return newfile
174 end
175

```

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

```

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

```

```

177 local name_e = "%f[^%a_]"
178 local btex_etex = name_b.."btex"..name_e.."s*(.)%s*"..name_b.."etex"..name_e
179 local verbatimetex_etex = name_b.."verbatimetex"..name_e.."s*(.)%s*"..name_b.."etex"..name_e
180
181 local function replaceinputmpfile (name,file)
182   local ofmodify = lfsattributes(file,"modification")
183   if not ofmodify then return file end
184   local newfile = name:gsub("%W","_")
185   newfile = format("%s/luamplib_input_%s", cachedir or outputdir, newfile)
186   if newfile and luamplibtime then
187     local nf = lfsattributes(newfile)
188     if nf and nf.mode == "file" and
189       ofmodify == nf.modification and luamplibtime < nf.access then
190       return nf.size == 0 and file or newfile
191     end
192   end
193
194   if name == "format.mp" then return replaceformatmp(file,newfile,ofmodify) end
195
196   local fh = ioopen(file,"r")
197   if not fh then return file end
198   local data = fh:read("*all"); fh:close()
199

```

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

```

200 local count,cnt = 0,0
201 data, cnt = data:gsub(btex_etex, "btex %1 etex ") -- space
202 count = count + cnt
203 data, cnt = data:gsub(verbatimetex_etex, "verbatimetex %1 etex;") -- semicolon
204 count = count + cnt
205
206 if count == 0 then
207   noneedtoreplace[name] = true
208   fh = ioopen(newfile,"w");
209   if fh then
210     fh:close()
211     lfstouch(newfile,currenttime,ofmodify)
212   end
213   return file
214 end
215
216 fh = ioopen(newfile,"w")
217 if not fh then return file end
218 fh:write(data); fh:close()
219 lfstouch(newfile,currenttime,ofmodify)
220 return newfile
221 end
222

```

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.

```

223 local mpkpse
224 do
225   local exe = 0

```

```

226 while arg[exe-1] do
227   exe = exe-1
228 end
229 mpkpse = kpse.new(arg[exe], "mpost")
230 end
231
232 local special_ftype = {
233   pfb = "type1 fonts",
234   enc = "enc files",
235 }
236
237 function luamplib.finder (name, mode, ftype)
238   if mode == "w" then
239     if name and name ~= "mpout.log" then
240       kpse.record_output_file(name) -- recorder
241     end
242     return name
243   else
244     ftype = special_ftype[ftype] or ftype
245     local file = mpkpse.find_file(name, ftype)
246     if file then
247       if lfstouch and ftype == "mp" and not noneedtoreplace[name] then
248         file = replaceinputmpfile(name, file)
249       end
250     else
251       file = mpkpse.find_file(name, name:match("%a+$"))
252     end
253     if file then
254       kpse.record_input_file(file) -- recorder
255     end
256     return file
257   end
258 end
259

```

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.)

```

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

```

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

```

267 local currentformat = "plain"
268 function luamplib.setformat (name)
269   currentformat = name
270 end
271

```

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

```

272 luamplib.codeinherit = false

```

```

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

```

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

```

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

```

296     local show = log:match"\n>>? .+"
297     if show then
298       termorlog("term", show, "Info (more info in the log)")
299       info(log)
300     elseif luamplib.showlog and log:find"%g" then
301       info(log)
302     end
303   end
304   return log
305 end
306 end
307
308 local function luamplibload (name)
309   local mpx = mplib.new {
310     ini_version = true,
311     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>.

```

312   make_text   = luamplib.maketext,
313   run_script  = luamplib.runscript,
314   math_mode   = luamplib.numbersystem,
315   job_name    = tex.jobname,
316   random_seed = math.random(4095),

```

```

317     extensions = 1,
318 }

```

Append our own MetaPost preamble to the preamble above.

```

319 local preamble = tableconcat{
320     format(preamble, replacesuffix(name,"mp")),
321     luamplib.preambles.mplibcode,
322     luamplib.legacy_verbatimtex and luamplib.preambles.legacyverbatim or "",
323     luamplib.texttextlabel and luamplib.preambles.texttextlabel or "",
324 }
325 local result, log
326 if not mpx then
327     result = { status = 99, error = "out of memory"}
328 else
329     result = mpx:execute(preamble)
330 end
331 log = reporterror(result)
332 return mpx, result, log
333 end
334

```

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

```

335 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

```

```

336 local currfmt
337 if instancename and instancename ~= "" then
338     currfmt = instancename
339     has_instancename = true
340 else
341     currfmt = tableconcat{
342         currentformat,
343         luamplib.numbersystem or "scaled",
344         tostring(luamplib.texttextlabel),
345         tostring(luamplib.legacy_verbatimtex),
346     }
347     has_instancename = false
348 end
349 local mpx = mplibinstances[currfmt]
350 local standalone = not (has_instancename or luamplib.codeinherit)
351 if mpx and standalone then
352     mpx:finish()
353 end
354 local log = ""
355 if standalone or not mpx then
356     mpx, _, log = luamplibload(currentformat)
357     mplibinstances[currfmt] = mpx
358 end
359 local converted, result = false, {}
360 if mpx and data then

```

```

361 result = mpx:execute(data)
362 local log = reporterror(result, log)
363 if log then
364   if result.fig then
365     converted = luamplib.convert(result)
366   end
367 end
368 else
369   err"Mem file unloadable. Maybe generated with a different version of mplib?"
370 end
371 return converted, result
372 end
373

```

dvipdfmx is supported, though nobody seems to use it.

```

374 local pdfmode = tex.outputmode > 0

```

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

```

375 local catlatex = luatexbase.registernumber("catcodetable@latex")
376 local catat11 = luatexbase.registernumber("catcodetable@atletter")
377

```

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

```

```

378 local function run_tex_code (str, cat)
379   texruntoks(function() texsprint(cat or catlatex, str) end)
380 end
381

```

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

```

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

```

For conversion of sp to bp.

```

383 local factor = 65536*(7227/7200)
384
385 local texttext_fmt = 'image(addto currentpicture doublepath unitsquare \z
386 xscaled %f yscaled %f shifted (0,-%f) \z
387 withprescript "mplibtextboxid=%i:%f:%f")'
388
389 local function process_tex_text (str)
390   if str then
391     local global = (has_instancename or luamplib.globaltexttext or luamplib.codeinherit)

```

```

392         and "\\global" or ""
393     local tex_box_id
394     if global == "" then
395         tex_box_id = texboxes.localid + 1
396         texboxes.localid = tex_box_id
397     else
398         local boxid = texboxes.globalid + 1
399         texboxes.globalid = boxid
400         run_tex_code(format(
401             [[\expandafter\newbox\csname luamplib.box.%s\endcsname]], boxid))
402         tex_box_id = tex.getcount'alloctionnumber'
403     end
404     run_tex_code(format("%s\\setbox%i\\hbox{%s}", global, tex_box_id, str))
405     local box = texgetbox(tex_box_id)
406     local wd = box.width / factor
407     local ht = box.height / factor
408     local dp = box.depth / factor
409     return textext_fmt:format(wd, ht+dp, dp, tex_box_id, wd, ht+dp)
410 end
411 return ""
412 end
413

```

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.

```

414 local mplibcolorfmt = {
415     xcolor = tableconcat{
416         [[\begingroup\let\XC@color\relax]],
417         [[\def\set@color{\global\mplibtmtoks\expandafter{\current@color}}]],
418         [[\color%s\endgroup]],
419     },
420     l3color = tableconcat{
421         [[\begingroup\def__color_select:N#1{\expandafter\__color_select:nn#1}]],
422         [[\def__color_backend_select:nn#1#2{\global\mplibtmtoks{#1 #2}}]],
423         [[\def__kernel_backend_literal:e#1{\global\mplibtmtoks\expandafter{\expanded{#1}}}],
424         [[\color_select:n%s\endgroup]],
425     },
426 }
427
428 local colfmt = is_defined'color_select:n' and "l3color" or "xcolor"
429 if colfmt == "l3color" then
430     run_tex_code{
431         "\\newcatcodetable\\luamplibcctabexplat",
432         "\\begingroup",
433         "\\catcode`@=11 ",
434         "\\catcode`_=11 ",
435         "\\catcode`:=11 ",
436         "\\savecatcodetable\\luamplibcctabexplat",
437         "\\endgroup",
438     }
439 end
440 local ccexplat = luatexbase.registernumber"luamplibcctabexplat"
441

```

```

442 local function process_color (str)
443   if str then
444     if not str:find("%b{") then
445       str = format("{%s}", str)
446     end
447     local myfmt = mplibcolorfmt[colfmt]
448     if colfmt == "l3color" and is_defined"color" then
449       if str:find("%b[") then
450         myfmt = mplibcolorfmt.xcolor
451       else
452         for _,v in ipairs(str:match"{{(.+)":explode"!") do
453           if not v:find("^%s*d+%s*$") then
454             local pp = get_macro(format("l__color_named_%s_prop",v))
455             if not pp or pp == "" then
456               myfmt = mplibcolorfmt.xcolor
457             break
458           end
459         end
460       end
461     end
462   end
463   run_tex_code(myfmt:format(str), ccexplat or catat11)
464   local t = texgettoks"mplibtmptoks"
465   if not pdfmode and not t:find"^pdf" then
466     t = t:gsub("%a+ (.+)", "pdf:bc [%1]")
467   end
468   return format('1 withprescript "mpliboverridecolor=%s"', t)
469 end
470 return ""
471 end
472
   for \mpdim or mplibdimen
473 local function process_dimen (str)
474   if str then
475     str = str:gsub"{{(.+)","%1"
476     run_tex_code(format([[ \mplibtmptoks \expandafter { \the \dimexpr %s \relax } ]], str))
477     return format("begingroup %s endgroup", texgettoks"mplibtmptoks")
478   end
479   return ""
480 end
481

```

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

```

482 local function process_verbatimtex_text (str)
483   if str then
484     run_tex_code(str)
485   end
486   return ""
487 end
488

```

For legacy verbatimtex process. verbatimtex ... etex before beginfig() is not ignored, but the T<sub>E</sub>X code is inserted just before the mplib box. And T<sub>E</sub>X code inside

beginfig() ... endfig is inserted after the mplib box.

```
489 local tex_code_pre_mplib = {}
490 luamplib.figid = 1
491 luamplib.in_the_fig = false
492
493 local function process_verbatimtex_prefig (str)
494   if str then
495     tex_code_pre_mplib[luamplib.figid] = str
496   end
497   return ""
498 end
499
500 local function process_verbatimtex_infig (str)
501   if str then
502     return format('special "postmplibverbtex=%s";', str)
503   end
504   return ""
505 end
506
507 local runscript_funcs = {
508   luamplibtext    = process_tex_text,
509   luamplibcolor   = process_color,
510   luamplibdimen   = process_dimen,
511   luamplibprefig  = process_verbatimtex_prefig,
512   luamplibinfig   = process_verbatimtex_infig,
513   luamplibverbtex = process_verbatimtex_text,
514 }
515
```

For metafun format. see issue #79.

```
516 mp = mp or {}
517 local mp = mp
518 mp.mf_path_reset = mp.mf_path_reset or function() end
519 mp.mf_finish_saving_data = mp.mf_finish_saving_data or function() end
520 mp.report = mp.report or info
521
```

metafun 2021-03-09 changes crashes luamplib.

```
522 catcodes = catcodes or {}
523 local catcodes = catcodes
524 catcodes.numbers = catcodes.numbers or {}
525 catcodes.numbers.ctxcatcodes = catcodes.numbers.ctxcatcodes or catlatex
526 catcodes.numbers.texcatcodes = catcodes.numbers.texcatcodes or catlatex
527 catcodes.numbers.luacatcodes = catcodes.numbers.luacatcodes or catlatex
528 catcodes.numbers.notcatcodes = catcodes.numbers.notcatcodes or catlatex
529 catcodes.numbers.vrbcatcodes = catcodes.numbers.vrbcatcodes or catlatex
530 catcodes.numbers.prtcacodes = catcodes.numbers.prtcacodes or catlatex
531 catcodes.numbers.txtcatcodes = catcodes.numbers.txtcatcodes or catlatex
532
```

A function from ConT<sub>E</sub>Xt general.

```
533 local function mpprint(buffer,...)
534   for i=1,select("#",...) do
535     local value = select(i,...)
536     if value ~= nil then
```

```

537     local t = type(value)
538     if t == "number" then
539         buffer[#buffer+1] = format("%.16f",value)
540     elseif t == "string" then
541         buffer[#buffer+1] = value
542     elseif t == "table" then
543         buffer[#buffer+1] = "(" .. tableconcat(value,",") .. ")"
544     else -- boolean or whatever
545         buffer[#buffer+1] = tostring(value)
546     end
547 end
548 end
549 end
550
551 function luamplib.runscript (code)
552     local id, str = code:match("(.-){(.*)}")
553     if id and str then
554         local f = runscript_funcs[id]
555         if f then
556             local t = f(str)
557             if t then return t end
558         end
559     end
560     local f = loadstring(code)
561     if type(f) == "function" then
562         local buffer = {}
563         function mp.print(...)
564             mpprint(buffer,...)
565         end
566         local res = {f()}
567         buffer = tableconcat(buffer)
568         if buffer and buffer ~= "" then
569             return buffer
570         end
571         buffer = {}
572         mpprint(buffer, table.unpack(res))
573         return tableconcat(buffer)
574     end
575     return ""
576 end
577
578     make_text must be one liner, so comment sign is not allowed.
579 local function protecttexcontents (str)
580     return str:gsub("\\%", "\\0PerCent\0")
581           :gsub("%%.\n", "")
582           :gsub("%%.\$", "")
583           :gsub("%zPerCent%", "\\%")
584           :gsub("%s+", " ")
585 end
586 luamplib.legacy_verbatimtex = true
587
588 function luamplib.maketext (str, what)
589     if str and str ~= "" then

```

```

590 str = protecttexcontents(str)
591 if what == 1 then
592   if not str:find("\\documentclass"..name_e) and
593     not str:find("\\begin%s*{document}") and
594     not str:find("\\documentstyle"..name_e) and
595     not str:find("\\usepackage"..name_e) then
596     if luamplib.legacy_verbatimtex then
597       if luamplib.in_the_fig then
598         return process_verbatimtex_infig(str)
599       else
600         return process_verbatimtex_prefig(str)
601       end
602     else
603       return process_verbatimtex_text(str)
604     end
605   end
606 else
607   return process_tex_text(str)
608 end
609 end
610 return ""
611 end
612
  luamplib's metapost color operators
613 local function colorsplit (res)
614   local t, tt = { }, res:gsub("[%[%]]", ""):explode()
615   local be = tt[1]:find"%d" and 1 or 2
616   for i=be, #tt do
617     if tt[i]:find"%a" then break end
618     t[#t+1] = tt[i]
619   end
620   return t
621 end
622
623 luamplib.gettexcolor = function (str, rgb)
624   local res = process_color(str):match"mpliboverridecolor=(.+)""
625   if res:find" cs " or res:find"@pdf.obj" then
626     if not rgb then
627       warn("%s is a spot color. Forced to CMYK", str)
628     end
629     run_tex_code({
630       "\\color_export:nnN{",
631       str,
632       "}{" ,
633       rgb and "space-sep-rgb" or "space-sep-cmyk",
634       "\\mplib_@tempa",
635     }, ccexplat)
636     return get_macro"mplib_@tempa":explode()
637   end
638   local t = colorsplit(res)
639   if #t == 3 or not rgb then return t end
640   if #t == 4 then
641     return { 1 - math.min(1,t[1]+t[4]), 1 - math.min(1,t[2]+t[4]), 1 - math.min(1,t[3]+t[4]) }
642   end

```

```

643 return { t[1], t[1], t[1] }
644 end
645
646 luamplib.shadecolor = function (str)
647 local res = process_color(str):match'"mpliboverridecolor=(.+)"'
648 if res:find" cs " or res:find"@pdf.obj" then -- spot color shade: 13 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}

```

```

649 run_tex_code({
650   [[\color_export:nnN{]], str, [[]{backend}\mplib_@tempa]],
651 },ccexplat)
652 local name = get_macro'mplib_@tempa':match'{{(-)}{.+}'
653 local t, obj = res:explode()
654 if pdfmode then
655   obj = t[1]:match"^(.+)"
656   if ltx.pdf and ltx.pdf.object_id then
657     obj = format("%s 0 R", ltx.pdf.object_id(obj))
658   else
659     run_tex_code({
660       [[\edef\mplib_@tempa{\pdf_object_ref:n{]], obj, "}}",
661       },ccexplat)
662     obj = get_macro'mplib_@tempa'
663   end
664 else
665   obj = t[2]
666 end
667 local value = t[3]:match"%[(-)%]" or t[3]
668 return format('(%s) withprescript"mplib_spotcolor=%s:%s"', value,obj,name)
669 end
670 return colorsplit(res)
671 end
672

```

#### luamplib's mplibgraphicstext operator

```

673 local running = -1073741824
674 local emboldenfonts = { }
675 local function getemboldenwidth (curr, fakebold)
676   local width = emboldenfonts.width
677   if not width then
678     local f
679     local function getglyph(n)
680       while n do
681         if n.head then
682           getglyph(n.head)
683         elseif n.font and n.font > 0 then
684           f = n.font; break
685         end
686         n = node.getnext(n)
687       end
688     end
689     getglyph(curr)
690     width = font.getcopy(f or font.current()).size * fakebold / factor * 10
691     emboldenfonts.width = width
692   end
693   return width
694 end
695 local function getrulewhatsit (line, wd, ht, dp)
696   line, wd, ht, dp = line/1000, wd/factor, ht/factor, dp/factor
697   local pl
698   local fmt = "%f w %f %f %f %f re %s"
699   if pdfmode then
700     pl = node.new("whatsit","pdf_literal")

```

```

701   pl.mode = 0
702   else
703     fmt = "pdf:content " .. fmt
704     pl = node.new("whatsit", "special")
705   end
706   pl.data = fmt:format(line, 0, -dp, wd, ht+dp, "B")
707   local ss = node.new"glue"
708   node.setglue(ss, 0, 65536, 65536, 2, 2)
709   pl.next = ss
710   return pl
711 end
712 local function getrulemetric (box, curr, bp)
713   local wd,ht,dp = curr.width, curr.height, curr.depth
714   wd = wd == running and box.width or wd
715   ht = ht == running and box.height or ht
716   dp = dp == running and box.depth or dp
717   if bp then
718     return wd/factor, ht/factor, dp/factor
719   end
720   return wd, ht, dp
721 end
722 local function embolden (box, curr, fakebold)
723   local head = curr
724   while curr do
725     if curr.head then
726       curr.head = embolden(curr, curr.head, fakebold)
727     elseif curr.replace then
728       curr.replace = embolden(box, curr.replace, fakebold)
729     elseif curr.leader then
730       if curr.leader.head then
731         curr.leader.head = embolden(curr.leader, curr.leader.head, fakebold)
732       elseif curr.leader.id == node.id"rule" then
733         local glue = node.effective_glue(curr, box)
734         local line = getemboldenwidth(curr, fakebold)
735         local wd,ht,dp = getrulemetric(box, curr.leader)
736         if box.id == node.id"hlist" then
737           wd = glue
738         else
739           ht, dp = 0, glue
740         end
741         local pl = getrulewhatsit(line, wd, ht, dp)
742         local pack = box.id == node.id"hlist" and node.hpack or node.vpack
743         local list = pack(pl, glue, "exactly")
744         head = node.insert_after(head, curr, list)
745         head, curr = node.remove(head, curr)
746       end
747     elseif curr.id == node.id"rule" and curr.subtype == 0 then
748       local line = getemboldenwidth(curr, fakebold)
749       local wd,ht,dp = getrulemetric(box, curr)
750       if box.id == node.id"vlist" then
751         ht, dp = 0, ht+dp
752       end
753       local pl = getrulewhatsit(line, wd, ht, dp)
754       local list

```

```

755     if box.id == node.id"hlist" then
756         list = node.hpack(pl, wd, "exactly")
757     else
758         list = node.vpack(pl, ht+dp, "exactly")
759     end
760     head = node.insert_after(head, curr, list)
761     head, curr = node.remove(head, curr)
762 elseif curr.id == node.id"glyph" and curr.font > 0 then
763     local f = curr.font
764     local i = emboldenfonts[f]
765     if not i then
766         local ft = font.getfont(f) or font.getcopy(f)
767         if pdfmode then
768             width = ft.size * fakebold / factor * 10
769             emboldenfonts.width = width
770             ft.mode, ft.width = 2, width
771             i = font.define(ft)
772         else
773             if ft.format ~= "opentype" and ft.format ~= "truetype" then
774                 goto skip_type1
775             end
776             local name = ft.name:gsub("'",''):gsub(';','$','')
777             name = format('%s;embolden=%s;',name,fakebold)
778             _, i = fonts.constructors.readanddefine(name,ft.size)
779         end
780         emboldenfonts[f] = i
781     end
782     curr.font = i
783 end
784 ::skip_type1::
785 curr = node.getnext(curr)
786 end
787 return head
788 end
789 local function graphictextcolor (col, filldraw)
790     if col:find"^[%d%.:]+$" then
791         col = col:explode":"
792         if pdfmode then
793             local op = #col == 4 and "k" or #col == 3 and "rg" or "g"
794             col[#col+1] = filldraw == "fill" and op or op:upper()
795             return tableconcat(col, " ")
796         end
797         return format("[%s]", tableconcat(col, " "))
798     end
799     col = process_color(col):match"mpliboverridecolor=(.+)"
800     if pdfmode then
801         local t, tt = col:explode(), { }
802         local b = filldraw == "fill" and 1 or #t/2+1
803         local e = b == 1 and #t/2 or #t
804         for i=b,e do
805             tt[#tt+1] = t[i]
806         end
807         return tableconcat(tt, " ")
808     end

```

```

809 return col:gsub("^.- ", "")
810 end
811 luamplib.graphicstext = function (text, fakebold, fc, dc)
812 local fmt = process_tex_text(text):sub(1,-2)
813 local id = tonumber(fmt:match"mplibtexboxid=(%d+):")
814 emboldenfonts.width = nil
815 local box = texgetbox(id)
816 box.head = embolden(box, box.head, fakebold)
817 local fill = graphicstextcolor(fc,"fill")
818 local draw = graphicstextcolor(dc,"draw")
819 local bc = pdfmode and "" or "pdf:bc "
820 return format('%s withprescript "mpliboverridecolor=%s%s %s"', fmt, bc, fill, draw)
821 end
822
      luamplib's mplibglyph operator
823 local function mperr (str)
824 return format("hide(errmessage %q)", str)
825 end
826 local function getangle (a,b,c)
827 local r = math.deg(math.atan(c.y-b.y, c.x-b.x) - math.atan(b.y-a.y, b.x-a.x))
828 if r > 180 then
829 r = r - 360
830 elseif r < -180 then
831 r = r + 360
832 end
833 return r
834 end
835 local function turning (t)
836 local r, n = 0, #t
837 for i=1,2 do
838 tableinsert(t, t[i])
839 end
840 for i=1,n do
841 r = r + getangle(t[i], t[i+1], t[i+2])
842 end
843 return r/360
844 end
845 local function glyphimage(t, fmt)
846 local q,p,r = {},{}
847 for i,v in ipairs(t) do
848 local cmd = v[#v]
849 if cmd == "m" then
850 p = {format('(%s,%s)',v[1],v[2])}
851 r = {{x=v[1],y=v[2]}}
852 else
853 local nt = t[i+1]
854 local last = not nt or nt[#nt] == "m"
855 if cmd == "l" then
856 local pt = t[i-1]
857 local seco = pt[#pt] == "m"
858 if (last or seco) and r[1].x == v[1] and r[1].y == v[2] then
859 else
860 tableinsert(p, format('--(%s,%s)',v[1],v[2]))
861 tableinsert(r, {x=v[1],y=v[2]})

```

```

862     end
863     if last then
864         tableinsert(p, '--cycle')
865     end
866 elseif cmd == "c" then
867     tableinsert(p, format('..controls(%s,%s)and(%s,%s)',v[1],v[2],v[3],v[4]))
868     if last and r[1].x == v[5] and r[1].y == v[6] then
869         tableinsert(p, '..cycle')
870     else
871         tableinsert(p, format('..(%s,%s)',v[5],v[6]))
872         if last then
873             tableinsert(p, '--cycle')
874         end
875         tableinsert(r, {x=v[5],y=v[6]})
876     end
877 else
878     return mperr"unknown operator"
879 end
880 if last then
881     tableinsert(q[ turning(r) > 0 and 1 or 2 ], tableconcat(p))
882 end
883 end
884 end
885 r = { }
886 if fmt == "opentype" then
887     for _,v in ipairs(q[1]) do
888         tableinsert(r, format('addto currentpicture contour %s;',v))
889     end
890     for _,v in ipairs(q[2]) do
891         tableinsert(r, format('addto currentpicture contour %s withcolor background;',v))
892     end
893 else
894     for _,v in ipairs(q[2]) do
895         tableinsert(r, format('addto currentpicture contour %s;',v))
896     end
897     for _,v in ipairs(q[1]) do
898         tableinsert(r, format('addto currentpicture contour %s withcolor background;',v))
899     end
900 end
901 return format('image(%s)', tableconcat(r))
902 end
903 if not table.tofile then require"lualibs-lpeg"; require"lualibs-table"; end
904 function luamplib.glyph (f, c)
905     local filename, subfont, instance, kind, shapedata
906     local fid = tonumber(f) or font.id(f)
907     if fid > 0 then
908         local fontdata = font.getfont(fid) or font.getcopy(fid)
909         filename, subfont, kind = fontdata.filename, fontdata.subfont, fontdata.format
910         instance = fontdata.specification and fontdata.specification.instance
911         filename = filename and filename:gsub("^harfloaded:", "")
912     else
913         local name
914         f = f:match"^%s*(.)%s*$"
915         name, subfont, instance = f:match"(.+)%((%d+)%)%[(.-)]%"

```

```

916   if not name then
917       name, instance = f:match"(.)%[(.-)%]$" -- SourceHanSansK-VF.otf[Heavy]
918   end
919   if not name then
920       name, subfont = f:match"(.)%((%d+)%)$" -- Times.ttc(2)
921   end
922   name = name or f
923   subfont = (subfont or 0)+1
924   instance = instance and instance:lower()
925   for _,ftype in ipairs{"opentype", "truetype"} do
926       filename = kpse.find_file(name, ftype.." fonts")
927       if filename then
928           kind = ftype; break
929       end
930   end
931 end
932 if kind ~= "opentype" and kind ~= "truetype" then
933     f = fid and fid > 0 and tex.fontname(fid) or f
934     if kpse.find_file(f, "tfm") then
935         return format("glyph %s of %q", tonumber(c) or format("%q",c), f)
936     else
937         return mperr"font not found"
938     end
939 end
940 local time = lfsattributes(filename,"modification")
941 local k = format("shapes_%s(%s)[%s]", filename, subfont or "", instance or "")
942 local h = format(string.rep('%02x', 256/8), string.byte(sha2.digest256(k), 1, -1))
943 local newname = format("%s/%s.lua", cachedir or outputdir, h)
944 local newtime = lfsattributes(newname,"modification") or 0
945 if time == newtime then
946     shapedata = require(newname)
947 end
948 if not shapedata then
949     shapedata = fonts and fonts.handlers.otf.readers.loadshapes(filename,subfont,instance)
950     if not shapedata then return mperr"loadshapes() failed. luaotfload not loaded?" end
951     table.tofile(newname, shapedata, "return")
952     lfstouch(newname, time, time)
953 end
954 local gid = tonumber(c)
955 if not gid then
956     local uni = utf8.codepoint(c)
957     for i,v in pairs(shapedata.glyphs) do
958         if c == v.name or uni == v.unicode then
959             gid = i; break
960         end
961     end
962 end
963 if not gid then return mperr"cannot get GID (glyph id)" end
964 local fac = 1000 / (shapedata.units or 1000)
965 local t = shapedata.glyphs[gid].segments
966 if not t then return "image(fill fullcircle scaled 0;)" end
967 for i,v in ipairs(t) do
968     if type(v) == "table" then
969         for ii,vv in ipairs(v) do

```

```

970     if type(vv) == "number" then
971         t[i][ii] = format("%.0f", vv * fac)
972     end
973 end
974 end
975 end
976 kind = shapedata.format or kind
977 return glyphimage(t, kind)
978 end
979
    mpliboutlinetext : based on mkiv's font-mps.lua
980 local rulefmt = "mpliboutlinepic[%i]:=image(addto currentpicture contour \z
981 unitsquare shifted - center unitsquare;) xscaled %f yscaled %f shifted (%f,%f);"
982 local outline_horz, outline_vert
983 function outline_vert (res, box, curr, xshift, yshift)
984     local b2u = box.dir == "LTL"
985     local dy = (b2u and -box.depth or box.height)/factor
986     local ody = dy
987     while curr do
988         if curr.id == node.id"rule" then
989             local wd, ht, dp = getrulemetric(box, curr, true)
990             local hd = ht + dp
991             if hd ~= 0 then
992                 dy = dy + (b2u and dp or -ht)
993                 if wd ~= 0 and curr.subtype == 0 then
994                     res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+wd/2, yshift+dy+(ht-dp)/2)
995                 end
996                 dy = dy + (b2u and ht or -dp)
997             end
998         elseif curr.id == node.id"glue" then
999             local vwidth = node.effective_glue(curr,box)/factor
1000             if curr.leader then
1001                 local curr, kind = curr.leader, curr.subtype
1002                 if curr.id == node.id"rule" then
1003                     local wd = getrulemetric(box, curr, true)
1004                     if wd ~= 0 then
1005                         local hd = vwidth
1006                         local dy = dy + (b2u and 0 or -hd)
1007                         if hd ~= 0 and curr.subtype == 0 then
1008                             res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+wd/2, yshift+dy+hd/2)
1009                         end
1010                     end
1011                 elseif curr.head then
1012                     local hd = (curr.height + curr.depth)/factor
1013                     if hd <= vwidth then
1014                         local dy, n, iy = dy, 0, 0
1015                         if kind == 100 or kind == 103 then -- todo: gleaders
1016                             local ady = abs(ody - dy)
1017                             local ndy = math.ceil(ady / hd) * hd
1018                             local diff = ndy - ady
1019                             n = (vwidth-diff) // hd
1020                             dy = dy + (b2u and diff or -diff)
1021                         else
1022                             n = vwidth // hd

```

```

1023         if kind == 101 then
1024             local side = vwidth % hd / 2
1025             dy = dy + (b2u and side or -side)
1026         elseif kind == 102 then
1027             iy = vwidth % hd / (n+1)
1028             dy = dy + (b2u and iy or -iy)
1029         end
1030     end
1031     dy = dy + (b2u and curr.depth or -curr.height)/factor
1032     hd = b2u and hd or -hd
1033     iy = b2u and iy or -iy
1034     local func = curr.id == node.id"hlist" and outline_horz or outline_vert
1035     for i=1,n do
1036         res = func(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
1037         dy = dy + hd + iy
1038     end
1039 end
1040 end
1041 end
1042 dy = dy + (b2u and vwidth or -vwidth)
1043 elseif curr.id == node.id"kern" then
1044     dy = dy + curr.kern/factor * (b2u and 1 or -1)
1045 elseif curr.id == node.id"vlist" then
1046     dy = dy + (b2u and curr.depth or -curr.height)/factor
1047     res = outline_vert(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
1048     dy = dy + (b2u and curr.height or -curr.depth)/factor
1049 elseif curr.id == node.id"hlist" then
1050     dy = dy + (b2u and curr.depth or -curr.height)/factor
1051     res = outline_horz(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
1052     dy = dy + (b2u and curr.height or -curr.depth)/factor
1053 end
1054 curr = node.getnext(curr)
1055 end
1056 return res
1057 end
1058 function outline_horz (res, box, curr, xshift, yshift, discwd)
1059     local r2l = box.dir == "TRT"
1060     local dx = r2l and (discwd or box.width/factor) or 0
1061     local dirs = { { dir = r2l, dx = dx } }
1062     while curr do
1063         if curr.id == node.id"dir" then
1064             local sign, dir = curr.dir:match"(.)(...)"
1065             local level, newdir = curr.level, r2l
1066             if sign == "+" then
1067                 newdir = dir == "TRT"
1068                 if r2l ~= newdir then
1069                     local n = node.getnext(curr)
1070                     while n do
1071                         if n.id == node.id"dir" and n.level+1 == level then break end
1072                         n = node.getnext(n)
1073                     end
1074                     n = n or node.tail(curr)
1075                     dx = dx + node.rangedimensions(box, curr, n)/factor * (newdir and 1 or -1)
1076                 end

```

```

1077     dirs[level] = { dir = r2l, dx = dx }
1078   else
1079     local level = level + 1
1080     newdir = dirs[level].dir
1081     if r2l ~= newdir then
1082       dx = dirs[level].dx
1083     end
1084   end
1085   r2l = newdir
1086 elseif curr.char and curr.font and curr.font > 0 then
1087   local ft = font.getfont(curr.font) or font.getcopy(curr.font)
1088   local gid = ft.characters[curr.char].index or curr.char
1089   local scale = ft.size / factor / 1000
1090   local slant = (ft.slant or 0)/1000
1091   local extend = (ft.extend or 1000)/1000
1092   local squeeze = (ft.squeeze or 1000)/1000
1093   local expand = 1 + (curr.expansion_factor or 0)/1000000
1094   local xscale = scale * extend * expand
1095   local yscale = scale * squeeze
1096   dx = dx - (r2l and curr.width/factor*expand or 0)
1097   local xpos = dx + xshift + (curr.xoffset or 0)/factor
1098   local ypos = yshift + (curr.yoffset or 0)/factor
1099   local vertical = ft.shared and ft.shared.features.vertical and "rotated 90" or ""
1100   if vertical ~= "" then -- luatexko
1101     for _,v in ipairs(ft.characters[curr.char].commands or { }) do
1102       if v[1] == "down" then
1103         ypos = ypos - v[2] / factor
1104       elseif v[1] == "right" then
1105         xpos = xpos + v[2] / factor
1106       else
1107         break
1108       end
1109     end
1110   end
1111   local image
1112   if ft.format == "opentype" or ft.format == "truetype" then
1113     image = luamplib.glyph(curr.font, gid)
1114   else
1115     local name, scale = ft.name, 1
1116     local vf = font.read_vf(name, ft.size)
1117     if vf and vf.characters[gid] then
1118       local cmds = vf.characters[gid].commands or {}
1119       for _,v in ipairs(cmds) do
1120         if v[1] == "char" then
1121           gid = v[2]
1122         elseif v[1] == "font" and vf.fonts[v[2]] then
1123           name = vf.fonts[v[2]].name
1124           scale = vf.fonts[v[2]].size / ft.size
1125         end
1126       end
1127     end
1128     image = format("glyph %s of %q scaled %f", gid, name, scale)
1129   end
1130   res[#res+1] = format("mpliboutlinepic[%i]:= %s xscaled %f yscaled %f slanted %f %s shifted (%f,%f);",

```

```

1131             #res+1, image, xscale, yscale, slant, vertical, xpos, ypos)
1132     dx = dx + (r2l and 0 or curr.width/factor*expand)
1133     elseif curr.replace then
1134         local width = node.dimensions(curr.replace)/factor
1135         dx = dx - (r2l and width or 0)
1136         res = outline_horz(res, box, curr.replace, xshift+dx, yshift, width)
1137         dx = dx + (r2l and 0 or width)
1138     elseif curr.id == node.id"rule" then
1139         local wd, ht, dp = getrulemetric(box, curr, true)
1140         if wd ~= 0 then
1141             local hd = ht + dp
1142             dx = dx - (r2l and wd or 0)
1143             if hd ~= 0 and curr.subtype == 0 then
1144                 res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
1145             end
1146             dx = dx + (r2l and 0 or wd)
1147         end
1148     elseif curr.id == node.id"glue" then
1149         local width = node.effective_glue(curr, box)/factor
1150         dx = dx - (r2l and width or 0)
1151         if curr.leader then
1152             local curr, kind = curr.leader, curr.subtype
1153             if curr.id == node.id"rule" then
1154                 local wd, ht, dp = getrulemetric(box, curr, true)
1155                 local hd = ht + dp
1156                 if hd ~= 0 then
1157                     wd = width
1158                     if wd ~= 0 and curr.subtype == 0 then
1159                         res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
1160                     end
1161                 end
1162             elseif curr.head then
1163                 local wd = curr.width/factor
1164                 if wd <= width then
1165                     local dx = r2l and dx+width or dx
1166                     local n, ix = 0, 0
1167                     if kind == 100 or kind == 103 then -- todo: gleaders
1168                         local adx = abs(dx-dirs[1].dx)
1169                         local ndx = math.ceil(adx / wd) * wd
1170                         local diff = ndx - adx
1171                         n = (width-diff) // wd
1172                         dx = dx + (r2l and -diff-wd or diff)
1173                     else
1174                         n = width // wd
1175                         if kind == 101 then
1176                             local side = width % wd / 2
1177                             dx = dx + (r2l and -side-wd or side)
1178                         elseif kind == 102 then
1179                             ix = width % wd / (n+1)
1180                             dx = dx + (r2l and -ix-wd or ix)
1181                         end
1182                     end
1183                 end
1184                 wd = r2l and -wd or wd
1185                 ix = r2l and -ix or ix

```

```

1185         local func = curr.id == node.id"hlist" and outline_horz or outline_vert
1186     for i=1,n do
1187         res = func(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1188         dx = dx + wd + ix
1189     end
1190 end
1191 end
1192 end
1193 dx = dx + (r2l and 0 or width)
1194 elseif curr.id == node.id"kern" then
1195     dx = dx + curr.kern/factor * (r2l and -1 or 1)
1196 elseif curr.id == node.id"math" then
1197     dx = dx + curr.surround/factor * (r2l and -1 or 1)
1198 elseif curr.id == node.id"vlist" then
1199     dx = dx - (r2l and curr.width/factor or 0)
1200     res = outline_vert(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1201     dx = dx + (r2l and 0 or curr.width/factor)
1202 elseif curr.id == node.id"hlist" then
1203     dx = dx - (r2l and curr.width/factor or 0)
1204     res = outline_horz(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1205     dx = dx + (r2l and 0 or curr.width/factor)
1206 end
1207 curr = node.getnext(curr)
1208 end
1209 return res
1210 end
1211 function luamplib.outlinetext (text)
1212     local fmt = process_tex_text(text)
1213     local id = tonumber(fmt:match"mplibtextboxid=(%d+):")
1214     local box = texgetbox(id)
1215     local res = outline_horz({ }, box, box.head, 0, 0)
1216     if #res == 0 then res = { "mpliboutlinepic[1]:=image(fill fullcircle scaled 0;);" } end
1217     return tableconcat(res) .. format("mpliboutlinenum:=%i;", #res)
1218 end
1219

```

### Our MetaPost preambles

```

1220 luamplib.preambles = {
1221     mplibcode = [[
1222 texscriptmode := 2;
1223 def rawtexttext (expr t) = runscript("luamplibtext{"&t&"}") enddef;
1224 def mplibcolor (expr t) = runscript("luamplibcolor{"&t&"}") enddef;
1225 def mplibdimen (expr t) = runscript("luamplibdimen{"&t&"}") enddef;
1226 def VerbatimTeX (expr t) = runscript("luamplibverbtex{"&t&"}") enddef;
1227 if known context_mlib:
1228     defaultfont := "cmtt10";
1229     let infont = normalinfont;
1230     let fontsize = normalfontsize;
1231     vardef thelabel@#(expr p,z) =
1232         if string p :
1233             thelabel@#(p infont defaultfont scaled defaultscale,z)
1234         else :
1235             p shifted (z + labeloffset*mfun_laboff@# -
1236                 (mfun_labxf@#*lrcorner p + mfun_labyf@#*ulcorner p +
1237                 (1-mfun_labxf@#-mfun_labyf@#)*llcorner p))

```

```

1238   fi
1239 enddef;
1240 else:
1241   vardef texttext@# (text t) = rawtexttext (t) enddef;
1242   def message expr t =
1243     if string t: runscript("mp.report[=["&t&"]=]") else: errmessage "Not a string" fi
1244   enddef;
1245 fi
1246 def resolvedcolor(expr s) =
1247   runscript("return luamplib.shadecolor('"& s &"')")
1248 enddef;
1249 def colordecimals primary c =
1250   if cmykcolor c:
1251     decimal cyanpart c & ":" & decimal magentapart c & ":" &
1252     decimal yellowpart c & ":" & decimal blackpart c
1253   elseif rgbcolor c:
1254     decimal redpart c & ":" & decimal greenpart c & ":" & decimal bluepart c
1255   elseif string c:
1256     if known graphicstextpic: c else: colordecimals resolvedcolor(c) fi
1257   else:
1258     decimal c
1259   fi
1260 enddef;
1261 def externalfigure primary filename =
1262   draw rawtexttext("\includegraphics{"& filename &}")
1263 enddef;
1264 def TEX = texttext enddef;
1265 def mplibtexcolor primary c =
1266   runscript("return luamplib.gettexcolor('"& c &"')")
1267 enddef;
1268 def mplibrbgtexcolor primary c =
1269   runscript("return luamplib.gettexcolor('"& c &"', 'rgb')")
1270 enddef;
1271 def mplibgraphicstext primary t =
1272   begingroup;
1273   mplibgraphicstext_ (t)
1274 enddef;
1275 def mplibgraphicstext_ (expr t) text rest =
1276   save fakebold, scale, fillcolor, drawcolor, withfillcolor, withdrawcolor,
1277   fb, fc, dc, graphicstextpic;
1278   picture graphicstextpic; graphicstextpic := nullpicture;
1279   numeric fb; string fc, dc; fb:=2; fc:="white"; dc:="black";
1280   let scale = scaled;
1281   def fakebold primary c = hide(fb:=c;) enddef;
1282   def fillcolor primary c = hide(fc:=colordecimals c;) enddef;
1283   def drawcolor primary c = hide(dc:=colordecimals c;) enddef;
1284   let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
1285   addto graphicstextpic doublepath origin rest; graphicstextpic:=nullpicture;
1286   def fakebold primary c = enddef;
1287   let fillcolor = fakebold; let drawcolor = fakebold;
1288   let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
1289   image(draw runscript("return luamplib.graphicstext(====["&t&"]====),"
1290     & decimal fb & ", ""& fc & ""', ""& dc & ""')") rest;)
1291 endgroup;

```

```

1292 endif;
1293 def mplibglyph expr c of f =
1294   runscript (
1295     "return luamplib.glyph('"
1296     & if numeric f: decimal fi f
1297     & "'',"
1298     & if numeric c: decimal fi c
1299     & "')"
1300   )
1301 endif;
1302 def mplibdrawglyph expr g =
1303   draw image(
1304     save i; numeric i; i:=0;
1305     for item within g:
1306       i := i+1;
1307       fill pathpart item
1308       if i < length g: withpostscript "collect" fi;
1309     endfor
1310   )
1311 endif;
1312 def mplib_do_outline_text_set_b (text f) (text d) text r =
1313   def mplib_do_outline_options_f = f endif;
1314   def mplib_do_outline_options_d = d endif;
1315   def mplib_do_outline_options_r = r endif;
1316 endif;
1317 def mplib_do_outline_text_set_f (text f) text r =
1318   def mplib_do_outline_options_f = f endif;
1319   def mplib_do_outline_options_r = r endif;
1320 endif;
1321 def mplib_do_outline_text_set_u (text f) text r =
1322   def mplib_do_outline_options_f = f endif;
1323 endif;
1324 def mplib_do_outline_text_set_d (text d) text r =
1325   def mplib_do_outline_options_d = d endif;
1326   def mplib_do_outline_options_r = r endif;
1327 endif;
1328 def mplib_do_outline_text_set_r (text d) (text f) text r =
1329   def mplib_do_outline_options_d = d endif;
1330   def mplib_do_outline_options_f = f endif;
1331   def mplib_do_outline_options_r = r endif;
1332 endif;
1333 def mplib_do_outline_text_set_n text r =
1334   def mplib_do_outline_options_r = r endif;
1335 endif;
1336 def mplib_do_outline_text_set_p = endif;
1337 def mplib_fill_outline_text =
1338   for n=1 upto mpliboutlinenum:
1339     i:=0;
1340     for item within mpliboutlinepic[n]:
1341       i:=i+1;
1342       fill pathpart item mplib_do_outline_options_f withpen pencircle scaled 0
1343       if (n<mpliboutlinenum) or (i<length mpliboutlinepic[n]): withpostscript "collect"; fi
1344     endfor
1345   endfor

```

```

1346 enddef;
1347 def mplib_draw_outline_text =
1348   for n=1 upto mpliboutlinenum:
1349     for item within mpliboutlinepic[n]:
1350       draw pathpart item mplib_do_outline_options_d;
1351     endfor
1352   endfor
1353 enddef;
1354 def mplib_filldraw_outline_text =
1355   for n=1 upto mpliboutlinenum:
1356     i:=0;
1357     for item within mpliboutlinepic[n]:
1358       i:=i+1;
1359       if (n<mpliboutlinenum) or (i<length mpliboutlinepic[n]):
1360         fill pathpart item mplib_do_outline_options_f withpostscript "collect";
1361       else:
1362         draw pathpart item mplib_do_outline_options_f withpostscript "both";
1363       fi
1364     endfor
1365   endfor
1366 enddef;
1367 vardef mpliboutlinetext@# (expr t) text rest =
1368   save kind; string kind; kind := str @#;
1369   save i; numeric i;
1370   picture mpliboutlinepic[]; numeric mpliboutlinenum;
1371   def mplib_do_outline_options_d = enddef;
1372   def mplib_do_outline_options_f = enddef;
1373   def mplib_do_outline_options_r = enddef;
1374   runscript("return luamplib.outlinetext[==["&t&"]==]");
1375   image ( addto currentpicture also image (
1376     if kind = "f":
1377       mplib_do_outline_text_set_f rest;
1378       mplib_fill_outline_text;
1379     elseif kind = "d":
1380       mplib_do_outline_text_set_d rest;
1381       mplib_draw_outline_text;
1382     elseif kind = "b":
1383       mplib_do_outline_text_set_b rest;
1384       mplib_fill_outline_text;
1385       mplib_draw_outline_text;
1386     elseif kind = "u":
1387       mplib_do_outline_text_set_u rest;
1388       mplib_filldraw_outline_text;
1389     elseif kind = "r":
1390       mplib_do_outline_text_set_r rest;
1391       mplib_draw_outline_text;
1392       mplib_fill_outline_text;
1393     elseif kind = "p":
1394       mplib_do_outline_text_set_p;
1395       mplib_draw_outline_text;
1396     else:
1397       mplib_do_outline_text_set_n rest;
1398       mplib_fill_outline_text;
1399     fi;

```

```

1400 ) mplib_do_outline_options_r; )
1401 enddef ;
1402 primarydef t withpattern p =
1403 image( fill t withprescript "mplibpattern=" & if numeric p: decimal fi p; )
1404 enddef;
1405 vardef mplibtransformmatrix (text e) =
1406 save t; transform t;
1407 t = inverse (identity e);
1408 runscript("luamplib.transformmatrix = {"
1409 & decimal xpart t & ","
1410 & decimal xypart t & ","
1411 & decimal ypart t & ","
1412 & decimal yxpart t & ","
1413 & decimal xpart t & ","
1414 & decimal ypart t & ","
1415 & "}");
1416 enddef;
1417 ]],
1418 legacyverbatim = [[
1419 def specialVerbatimTeX (text t) = runscript("luamplibprefig{"&t&}") enddef;
1420 def normalVerbatimTeX (text t) = runscript("luamplibinfig{"&t&}") enddef;
1421 let VerbatimTeX = specialVerbatimTeX;
1422 extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;"&
1423 "runscript(" &ditto& "luamplib.in_the_fig=true" &ditto& ");";
1424 extra_endfig := extra_endfig & " let VerbatimTeX = specialVerbatimTeX;"&
1425 "runscript(" &ditto&
1426 "if luamplib.in_the_fig then luamplib.figid=luamplib.figid+1 end "&
1427 "luamplib.in_the_fig=false" &ditto& ");";
1428 ]],
1429 texttextlabel = [[
1430 primarydef s infont f = rawtexttext(s) enddef;
1431 def fontsize expr f =
1432 begingroup
1433 save size; numeric size;
1434 size := mplibdimen("1em");
1435 if size = 0: 10pt else: size fi
1436 endgroup
1437 enddef;
1438 ]],
1439 }
1440

```

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

```

1441 luamplib.verbatiminput = false
1442

```

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

```

1443 local function protect_expansion (str)
1444 if str then
1445 str = str:gsub("\\", "!!!Control!!!")
1446 :gsub("%%", "!!!Comment!!!")
1447 :gsub("#", "!!!HashSign!!!")
1448 :gsub("{", "!!!LBrace!!!")
1449 :gsub("}", "!!!RBrace!!!")
1450 return format("\\unexpanded{%s}", str)

```

```

1451 end
1452 end
1453
1454 local function unprotect_expansion (str)
1455   if str then
1456     return str:gsub("!!!Control!!!", "\\")
1457           :gsub("!!!Comment!!!", "%")
1458           :gsub("!!!HashSign!!!", "#")
1459           :gsub("!!!LBrace!!!", "{")
1460           :gsub("!!!RBrace!!!", "}")
1461   end
1462 end
1463
1464 luamplib.everymplib = setmetatable({ [""] = "" },{ __index = function(t) return t[""] end })
1465 luamplib.everyendmplib = setmetatable({ [""] = "" },{ __index = function(t) return t[""] end })
1466
1467 function luamplib.process_mplibcode (data, instancename)
1468   texboxes.localid = 4096
1469

```

This is needed for legacy behavior

```

1470   if luamplib.legacy_verbatimmex then
1471     luamplib.figid, tex_code_pre_mplib = 1, {}
1472   end
1473
1474   local everymplib = luamplib.everymplib[instancename]
1475   local everyendmplib = luamplib.everyendmplib[instancename]
1476   data = format("\n%s\n%s\n%s\n", everymplib, data, everyendmplib)
1477   :gsub("\r", "\n")
1478

```

These five lines are needed for mplibverbatim mode.

```

1479   if luamplib.verbatiminput then
1480     data = data:gsub("\mpcolor%{.+}%{b}", "mplibcolor(\%1)")
1481           :gsub("\mpdim%{.+}%{b}", "mplibdimen(\%1)")
1482           :gsub("\mpdim%{.+}%{a+}", "mplibdimen(\%1)")
1483           :gsub(btex_etex, "btex %1 etex ")
1484           :gsub(verbatimmex_etex, "verbatimmex %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.

```

1485   else
1486     data = data:gsub(btex_etex, function(str)
1487       return format("btex %s etex ", protect_expansion(str)) -- space
1488     end)
1489     :gsub(verbatimmex_etex, function(str)
1490       return format("verbatimmex %s etex;", protect_expansion(str)) -- semicolon
1491     end)
1492     :gsub("\.-", "\", protect_expansion)
1493     :gsub("\%", "\0PerCent\0")
1494     :gsub("%%.-\n", "\n")
1495     :gsub("%zPerCentz", "\%")
1496     run_tex_code(format("\mplibmptoks\expandafter{\expanded{}}", data))
1497     data = texgettoks"mplibmptoks"

```

Next line to address issue #55

```

1498 :gsub("##", "#")
1499 :gsub("\\.-\\", unprotect_expansion)
1500 :gsub(btex_etex, function(str)
1501   return format("btex %s etex", unprotect_expansion(str))
1502 end)
1503 :gsub(verbatimetex, function(str)
1504   return format("verbatimetex %s etex", unprotect_expansion(str))
1505 end)
1506 end
1507
1508 process(data, instancename)
1509 end
1510

```

For parsing prescript materials.

```

1511 local further_split_keys = {
1512   mplibtexboxid = true,
1513   sh_color_a    = true,
1514   sh_color_b    = true,
1515 }
1516 local function script2table(s)
1517   local t = {}
1518   for _,i in ipairs(s:explode("\\13+")) do
1519     local k,v = i:match("(.-)=(.*)") -- v may contain = or empty.
1520     if k and v and k ~= "" and not t[k] then
1521       if further_split_keys[k] or further_split_keys[k:sub(1,10)] then
1522         t[k] = v:explode(":")
1523       else
1524         t[k] = v
1525       end
1526     end
1527   end
1528   return t
1529 end
1530

```

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

```

1531 local function getobjects(result, figure, f)
1532   return figure:objects()
1533 end
1534
1535 function luamplib.convert (result, flusher)
1536   luamplib.flush(result, flusher)
1537   return true -- done
1538 end
1539
1540 local figcontents = { post = { } }
1541 local function put2output(a,...)
1542   figcontents[#figcontents+1] = type(a) == "string" and format(a,...) or a
1543 end
1544
1545 local function pdf_startfigure(n,llx,lly,urx,ury)
1546   put2output("\\mplibstarttoPDF{%f}{%f}{%f}{%f}", llx, lly, urx, ury)
1547 end

```

```

1548
1549 local function pdf_stopfigure()
1550   put2output{"\mplibstoptoPDF"}
1551 end
1552
    tex.sprint with catcode regime -2, as sometimes # gets doubled in the argument of
pdfliteral.
1553 local function pdf_literalcode (fmt,...)
1554   put2output{-2, format(fmt,...)}
1555 end
1556
1557 local function pdf_textfigure(font,size,text,width,height,depth)
1558   text = text:gsub(".",function(c)
1559     return format("\hbox{\char%i}",string.byte(c)) -- kerning happens in metapost : false
1560   end)
1561   put2output{"\mplibtexttext{%s}{%f}{%s}{%s}",font,size,text,0,0}
1562 end
1563
1564 local bend_tolerance = 131/65536
1565
1566 local rx, sx, sy, ry, tx, ty, divider = 1, 0, 0, 1, 0, 0, 1
1567
1568 local function pen_characteristics(object)
1569   local t = mplib.pen_info(object)
1570   rx, ry, sx, sy, tx, ty = t.rx, t.ry, t.sx, t.sy, t.tx, t.ty
1571   divider = sx*sy - rx*ry
1572   return not (sx==1 and rx==0 and ry==0 and sy==1 and tx==0 and ty==0), t.width
1573 end
1574
1575 local function concat(px, py) -- no tx, ty here
1576   return (sy*px-ry*py)/divider,(sx*py-rx*px)/divider
1577 end
1578
1579 local function curved(ith,pth)
1580   local d = pth.left_x - ith.right_x
1581   if abs(ith.right_x - ith.x_coord - d) <= bend_tolerance and abs(pth.x_coord - pth.left_x - d) <= bend_tolerance then
1582     d = pth.left_y - ith.right_y
1583     if abs(ith.right_y - ith.y_coord - d) <= bend_tolerance and abs(pth.y_coord - pth.left_y - d) <= bend_tolerance then
1584       return false
1585     end
1586   end
1587   return true
1588 end
1589
1590 local function flushnormalpath(path,open)
1591   local pth, ith
1592   for i=1,#path do
1593     pth = path[i]
1594     if not ith then
1595       pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
1596     elseif curved(ith,pth) then
1597       pdf_literalcode("%f %f %f %f %f c",ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,pth.y_coord)
1598     else

```

```

1599     pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
1600   end
1601   ith = pth
1602 end
1603 if not open then
1604   local one = path[1]
1605   if curved(pth,one) then
1606     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 )
1607   else
1608     pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
1609   end
1610 elseif #path == 1 then -- special case .. draw point
1611   local one = path[1]
1612   pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
1613 end
1614 end
1615
1616 local function flushconcatpath(path,open)
1617 pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
1618 local pth, ith
1619 for i=1,#path do
1620   pth = path[i]
1621   if not ith then
1622     pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
1623   elseif curved(ith,pth) then
1624     local a, b = concat(ith.right_x,ith.right_y)
1625     local c, d = concat(pth.left_x,pth.left_y)
1626     pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
1627   else
1628     pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
1629   end
1630   ith = pth
1631 end
1632 if not open then
1633   local one = path[1]
1634   if curved(pth,one) then
1635     local a, b = concat(pth.right_x,pth.right_y)
1636     local c, d = concat(one.left_x,one.left_y)
1637     pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(one.x_coord, one.y_coord))
1638   else
1639     pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
1640   end
1641 elseif #path == 1 then -- special case .. draw point
1642   local one = path[1]
1643   pdf_literalcode("%f %f l",concat(one.x_coord,one.y_coord))
1644 end
1645 end
1646
1647 local function start_pdf_code()
1648 if pdfmode then
1649   pdf_literalcode("q")
1650 else
1651   put2output"\special{pdf:bcontent}"
1652 end

```

```

1653 end
1654 local function stop_pdf_code()
1655   if pdfmode then
1656     pdf_literalcode("Q")
1657   else
1658     put2output"\special{pdf:econtent}"
1659   end
1660 end
1661

```

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

```

1662 local function put_tex_boxes (object,prescript)
1663   local box = prescript.mplibtexboxid
1664   local n,tw,th = box[1],tonumber(box[2]),tonumber(box[3])
1665   if n and tw and th then
1666     local op = object.path
1667     local first, second, fourth = op[1], op[2], op[4]
1668     local tx, ty = first.x_coord, first.y_coord
1669     local sx, rx, ry, sy = 1, 0, 0, 1
1670     if tw ~= 0 then
1671       sx = (second.x_coord - tx)/tw
1672       rx = (second.y_coord - ty)/tw
1673       if sx == 0 then sx = 0.00001 end
1674     end
1675     if th ~= 0 then
1676       sy = (fourth.y_coord - ty)/th
1677       ry = (fourth.x_coord - tx)/th
1678       if sy == 0 then sy = 0.00001 end
1679     end
1680     start_pdf_code()
1681     pdf_literalcode("%f %f %f %f %f %f cm",sx,rx,ry,sy,tx,ty)
1682     put2output("\mplibputtextbox{%i}",n)
1683     stop_pdf_code()
1684   end
1685 end
1686

```

### Colors

```

1687 local prev_override_color
1688 local function do_preobj_CR(object,prescript)
1689   if object.postscript == "collect" then return end
1690   local override = prescript and prescript.mpliboverridecolor
1691   if override then
1692     if pdfmode then
1693       pdf_literalcode(override)
1694       override = nil
1695     else
1696       put2output("\special{%s}",override)
1697       prev_override_color = override
1698     end
1699   else
1700     local cs = object.color
1701     if cs and #cs > 0 then
1702       pdf_literalcode(luamplib.colorconverter(cs))

```

```

1703     prev_override_color = nil
1704 elseif not pdfmode then
1705     override = prev_override_color
1706     if override then
1707         put2output("\special{%s}",override)
1708     end
1709 end
1710 end
1711 return override
1712 end
1713
    For transparency and shading
1714 local pdfmanagement = is_defined'pdfmanagement_add:nnn'
1715 local pdfobjs, pdfetcs = {}, {}
1716 pdfetcs.pgftxtgs = "pgf@sys@addpdfresource@extgs@plain"
1717 pdfetcs.pgfpattern = "pgf@sys@addpdfresource@patterns@plain"
1718 pdfetcs.pgfcolorspace = "pgf@sys@addpdfresource@colorspaces@plain"
1719
1720 local function update_pdfobjs (os)
1721     local on = pdfobjs[os]
1722     if on then
1723         return on,false
1724     end
1725     if pdfmode then
1726         on = pdf.immediateobj(os)
1727     else
1728         on = pdfetcs.cnt or 1
1729         texsprintf(format("\special{pdf:obj @mplibpdfobj%s %s}",on,os))
1730         pdfetcs.cnt = on + 1
1731     end
1732     pdfobjs[os] = on
1733     return on,true
1734 end
1735
1736 if pdfmode then
1737 pdfetcs.getpageres = pdf.getpageresources or function() return pdf.pageresources end
1738 pdfetcs.setpageres = pdf.setpageresources or function(s) pdf.pageresources = s end
1739 pdfetcs.initialize_resources = function (name)
1740     local tabname = format("%s_res",name)
1741     pdfetcs[tabname] = { }
1742     if luatexbase.callbacktypes.finish_pdffile then -- ltuatex
1743         local obj = pdf.reserveobj()
1744         pdfetcs.setpageres(format("%s/%s %i 0 R", pdfetcs.getpageres() or "", name, obj))
1745         luatexbase.add_to_callback("finish_pdffile", function()
1746             pdf.immediateobj(obj, format("<<%s>>", tableconcat(pdfetcs[tabname])))
1747         end,
1748             format("luamplib.%s.finish_pdffile",name))
1749     end
1750 end
1751 pdfetcs.fallback_update_resources = function (name, res)
1752     if luatexbase.callbacktypes.finish_pdffile then
1753         local t = pdfetcs[format("%s_res",name)]
1754         t[#t+1] = res
1755     else

```

```

1756     local tpr, n = pdfetcs.getpageres() or "", 0
1757     tpr, n = tpr:gsub(format("/%s<<",name), "%1".res)
1758     if n == 0 then
1759         tpr = format("%s/%s<<%s>>", tpr, name, res)
1760     end
1761     pdfetcs.setpageres(tpr)
1762 end
1763 end
1764 else
1765     texsprint("\\special{pdf:obj @MPLibTr<<>>}", "\\special{pdf:obj @MPLibSh<<>>}",
1766 "\\special{pdf:obj @MPLibCS<<>>}", "\\special{pdf:obj @MPLibPt<<>>}")
1767 end
1768
    Transparency
1769 local transparency_modes = { [0] = "Normal",
1770 "Normal",      "Multiply",    "Screen",      "Overlay",
1771 "SoftLight",   "HardLight",   "ColorDodge", "ColorBurn",
1772 "Darken",     "Lighten",     "Difference",  "Exclusion",
1773 "Hue",        "Saturation",  "Color",      "Luminosity",
1774 "Compatible",
1775 }
1776
1777 local function update_tr_res(mode,opaque)
1778     local os = format("<</BM /%s/ca %.3f/CA %.3f/AIS false>>",mode,opaque,opaque)
1779     local on, new = update_pdfobjs(os)
1780     if not new then return on end
1781     local key = format("MPLibTr%s", on)
1782     local val = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s", on)
1783     if pdfmanagement then
1784         texsprint(ccexplat,
1785             format("\\pdfmanagement_add:nnn{Page/Resources/ExtGState}{%s}{%s}", key, val))
1786     else
1787         local tr = format("/%s %s", key, val)
1788         if is_defined(pdfetcs.pgfgextgs) then
1789             texsprint(format("\\csname %s\\endcsname{%s}", pdfetcs.pgfgextgs,tr))
1790         elseif pdfmode then
1791             if is_defined"TRP@list" then
1792                 texsprint(catat11,{
1793                     [[\if@files\immediate\write\@auxout{]],
1794                     [[\string\g@addto@macro\string\TRP@list{]],
1795                     tr,
1796                     [[]}\fi]],
1797                 })
1798             if not get_macro"TRP@list":find(tr) then
1799                 texsprint(catat11,[[\global\TRP@reruntrue]])
1800             end
1801         else
1802             if not pdfetcs.ExtGState_res then
1803                 pdfetcs.initialize_resources"ExtGState"
1804             end
1805             pdfetcs.fallback_update_resources("ExtGState", tr)
1806         end
1807     else
1808         texsprint(format("\\special{pdf:put @MPLibTr<<%s>>}",tr))

```

```

1809     texsprintf"\special{pdf:put @resources<</ExtGState @MPLibTr>>}"
1810   end
1811 end
1812 return on
1813 end
1814
1815 local function do_preobj_TR(object,prescript)
1816   if object.postscript == "collect" then return end
1817   local opa = prescript and prescript.tr_transparency
1818   local tron_no
1819   if opa then
1820     local mode = prescript.tr_alternative or 1
1821     mode = transparency_modes[tonumber(mode)]
1822     tron_no = update_tr_res(mode, opa)
1823     start_pdf_code()
1824     pdf_literalcode("/MPLibTr%i gs",tron_no)
1825   end
1826   return tron_no
1827 end
1828

```

#### Shading with metafun format.

```

1829 local function sh_pdfpageresources(shtype,domain,colorspace,ca,cb,coordinates,steps,fractions)
1830   local fun2fmt,os = "<</FunctionType 2/Domain [%s]/C0 [%s]/C1 [%s]/N 1>>"
1831   if steps > 1 then
1832     local list,bounds,encode = { },{ },{ }
1833     for i=1,steps do
1834       if i < steps then
1835         bounds[i] = fractions[i] or 1
1836       end
1837       encode[2*i-1] = 0
1838       encode[2*i] = 1
1839       os = fun2fmt:format(domain,tableconcat(ca[i], ' '),tableconcat(cb[i], ' '))
1840       list[i] = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s",update_pdfobjs(os))
1841     end
1842     os = tableconcat {
1843       "<</FunctionType 3",
1844       format("/Bounds [%s]", tableconcat(bounds, ' ')),
1845       format("/Encode [%s]", tableconcat(encode, ' ')),
1846       format("/Functions [%s]", tableconcat(list, ' ')),
1847       format("/Domain [%s]>>", domain),
1848     }
1849   else
1850     os = fun2fmt:format(domain,tableconcat(ca[1], ' '),tableconcat(cb[1], ' '))
1851   end
1852   local objref = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s",update_pdfobjs(os))
1853   os = tableconcat {
1854     format("<</ShadingType %i", shtype),
1855     format("/ColorSpace %s", colorspace),
1856     format("/Function %s", objref),
1857     format("/Coords [%s]", coordinates),
1858     "/Extend [true true]/AntiAlias true>>",
1859   }
1860   local on, new = update_pdfobjs(os)
1861   if not new then return on end

```

```

1862 local key = format("MPLibSh%s", on)
1863 local val = format(pdfmode and "%s 0 R" or "@mplibpdfobj%s", on)
1864 if pdfmanagement then
1865     texsprintf(ccexplat,
1866     format("\pdfmanagement_add:nnn{Page/Resources/Shading}{%s}{%s}", key, val))
1867 else
1868     local res = format("/%s %s", key, val)
1869     if pdfmode then
1870         if not pdfetcs.Shading_res then
1871             pdfetcs.initialize_resources"Shading"
1872         end
1873         pdfetcs.fallback_update_resources("Shading", res)
1874     else
1875         texsprintf(format("\special{pdf:put @MPLibSh<<%s>>}", res))
1876         texsprintf("\special{pdf:put @resources<</Shading @MPLibSh>>}")
1877     end
1878 end
1879 return on
1880 end
1881
1882 local function color_normalize(ca,cb)
1883     if #cb == 1 then
1884         if #ca == 4 then
1885             cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
1886         else -- #ca = 3
1887             cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
1888         end
1889     elseif #cb == 3 then -- #ca == 4
1890         cb[1], cb[2], cb[3], cb[4] = 1-cb[1], 1-cb[2], 1-cb[3], 0
1891     end
1892 end
1893
1894 pdfetcs.clrspcs = setmetatable({ }, { __index = function(t,names)
1895     run_tex_code({
1896         [[\color_model_new:nnn]],
1897         format("{mplibcolorspace_%s}", names:gsub(",","_")),
1898         format("{DeviceN}{names={%s}}", names),
1899         [[\edef\mplib@tempa{\pdf_object_ref_last:}]],
1900     }, ccexplat)
1901     local colorspace = get_macro'mplib@tempa'
1902     t[names] = colorspace
1903     return colorspace
1904 end })
1905
1906 local function do_preobj_SH(object,prescript)
1907     local shade_no
1908     local sh_type = prescript and prescript.sh_type
1909     if not sh_type then
1910         return
1911     else
1912         local domain = prescript.sh_domain or "0 1"
1913         local centera = prescript.sh_center_a or "0 0"; centera = centera:explode()
1914         local centerb = prescript.sh_center_b or "0 0"; centerb = centerb:explode()
1915         local transform = prescript.sh_transform == "yes"

```

```

1916 local sx,sy,sr,dx,dy = 1,1,1,0,0
1917 if transform then
1918     local first = prescript.sh_first or "0 0"; first = first:explode()
1919     local setx = prescript.sh_set_x or "0 0"; setx = setx:explode()
1920     local sety = prescript.sh_set_y or "0 0"; sety = sety:explode()
1921     local x,y = tonumber(setx[1]) or 0, tonumber(sety[1]) or 0
1922     if x ~= 0 and y ~= 0 then
1923         local path = object.path
1924         local path1x = path[1].x_coord
1925         local path1y = path[1].y_coord
1926         local path2x = path[x].x_coord
1927         local path2y = path[y].y_coord
1928         local dxa = path2x - path1x
1929         local dya = path2y - path1y
1930         local dxb = setx[2] - first[1]
1931         local dyb = sety[2] - first[2]
1932         if dxa ~= 0 and dya ~= 0 and dxb ~= 0 and dyb ~= 0 then
1933             sx = dxa / dxb ; if sx < 0 then sx = - sx end
1934             sy = dya / dyb ; if sy < 0 then sy = - sy end
1935             sr = math.sqrt(sx^2 + sy^2)
1936             dx = path1x - sx*first[1]
1937             dy = path1y - sy*first[2]
1938         end
1939     end
1940 end
1941 local ca, cb, colorspace, steps, fractions
1942 ca = { prescript.sh_color_a_1 or prescript.sh_color_a or {} }
1943 cb = { prescript.sh_color_b_1 or prescript.sh_color_b or {} }
1944 steps = tonumber(prescript.sh_step) or 1
1945 if steps > 1 then
1946     fractions = { prescript.sh_fraction_1 or 0 }
1947     for i=2,steps do
1948         fractions[i] = prescript[format("sh_fraction_%i",i)] or (i/steps)
1949         ca[i] = prescript[format("sh_color_a_%i",i)] or {}
1950         cb[i] = prescript[format("sh_color_b_%i",i)] or {}
1951     end
1952 end
1953 if prescript.mplib_spotcolor then
1954     ca, cb = { }, { }
1955     local names, pos, objref = { }, -1, ""
1956     local script = object.prescript:explode"\13+"
1957     for i=#script,1,-1 do
1958         if script[i]:find"mplib_spotcolor" then
1959             local name, value
1960             objref, name = script[i]:match"=(.-):(.)"
1961             value = script[i+1]:match"=(.*)"
1962             if not names[name] then
1963                 pos = pos+1
1964                 names[name] = pos
1965                 names[#names+1] = name
1966             end
1967             local t = { }
1968             for j=1,names[name] do t[#t+1] = 0 end
1969             t[#t+1] = value

```

```

1970         tableinsert(#ca == #cb and ca or cb, t)
1971     end
1972 end
1973 for _,t in ipairs{ca,cb} do
1974     for _,tt in ipairs(t) do
1975         for i=1,#names-#tt do tt[#tt+1] = 0 end
1976     end
1977 end
1978 if #names == 1 then
1979     colorspace = objref
1980 else
1981     colorspace = pdfetcs.clrspcs[ tableconcat(names,",") ]
1982 end
1983 else
1984     local model = 0
1985     for _,t in ipairs{ca,cb} do
1986         for _,tt in ipairs(t) do
1987             model = model > #tt and model or #tt
1988         end
1989     end
1990     for _,t in ipairs{ca,cb} do
1991         for _,tt in ipairs(t) do
1992             if #tt < model then
1993                 color_normalize(model == 4 and {1,1,1,1} or {1,1,1},tt)
1994             end
1995         end
1996     end
1997     colorspace = model == 4 and "/DeviceCMYK"
1998                 or model == 3 and "/DeviceRGB"
1999                 or model == 1 and "/DeviceGray"
2000                 or err"unknown color model"
2001 end
2002 if sh_type == "linear" then
2003     local coordinates = format("%f %f %f %f",
2004         dx + sx*centera[1], dy + sy*centera[2],
2005         dx + sx*centerb[1], dy + sy*centerb[2])
2006     shade_no = sh_pdfpageresources(2,domain,colorspace,ca,cb,coordinates,steps,fractions)
2007 elseif sh_type == "circular" then
2008     local factor = prescript.sh_factor or 1
2009     local radiusa = factor * prescript.sh_radius_a
2010     local radiusb = factor * prescript.sh_radius_b
2011     local coordinates = format("%f %f %f %f %f %f",
2012         dx + sx*centera[1], dy + sy*centera[2], sr*radiusa,
2013         dx + sx*centerb[1], dy + sy*centerb[2], sr*radiusb)
2014     shade_no = sh_pdfpageresources(3,domain,colorspace,ca,cb,coordinates,steps,fractions)
2015 else
2016     err"unknown shading type"
2017 end
2018 pdf_literalcode("q /Pattern cs")
2019 end
2020 return shade_no
2021 end
2022

```

## Patterns

```
2023 patterns = { }
2024 function luamplib.registerpattern ( boxid, name, opts )
2025   local box = texgetbox(boxid)
2026   local wd = format("%.3f",box.width/factor)
2027   local hd = format("%.3f",(box.height+box.depth)/factor)
2028   info("w/h/d of '%s': %s %s 0.0", name, wd, hd)
2029   if opts.xstep == 0 then opts.xstep = nil end
2030   if opts.ystep == 0 then opts.ystep = nil end
2031   if opts.colored == nil then opts.colored = true end
2032   if type(opts.matrix) == "table" then opts.matrix = tableconcat(opts.matrix," ") end
2033   if type(opts.bbox) == "table" then opts.bbox = tableconcat(opts.bbox," ") end
2034   if opts.matrix and opts.matrix:find"%a" then
2035     local data = format("mplibtransformmatrix(%s);",opts.matrix)
2036     process(data,"@mplibtransformmatrix")
2037     local t = luamplib.transformmatrix
2038     opts.matrix = format("%s %s %s %s", t[1], t[2], t[3], t[4])
2039     opts.xshift = opts.xshift or t[5]
2040     opts.yshift = opts.yshift or t[6]
2041   end
2042   local attr = {
2043     "/Type/Pattern",
2044     "/PatternType 1",
2045     format("/PaintType %i", opts.colored and 1 or 2),
2046     "/TilingType 2",
2047     format("/XStep %s", opts.xstep or wd),
2048     format("/YStep %s", opts.ystep or hd),
2049     format("/Matrix [%s %s %s]", opts.matrix or "1 0 0 1", opts.xshift or 0, opts.yshift or 0),
2050   }
2051   if pdfmode then
2052     local optres, t = opts.resources or "", { }
2053     if pdfmanagement then
2054       for _,v in ipairs{"ExtGState","ColorSpace","Shading"} do
2055         local pp = get_macro(format("g__pdfdict_/g__pdf_Core/Page/Resources/%s_prop",v))
2056         if pp and pp:find"__prop_pair" then
2057           t[#t+1] = format("/%s %s", v, ltx.__pdf.object["__pdf/Page/Resources"/..v])
2058         end
2059       end
2060     else
2061       local res = pdfetcs.getpageres() or ""
2062       run_tex_code[["\mplibtmptoks\expandafter{\the\pdfvariable pageresources}]]
2063       res = (res .. texgettoks'mplibtmptoks'):explode()
2064       res = tableconcat(res," "):explode"/+"
2065       for _,v in ipairs(res) do
2066         if not v:find"Pattern" and not optres:find(v) then
2067           t[#t+1] = "/" .. v
2068         end
2069       end
2070     end
2071     optres = optres .. tableconcat(t)
2072     if opts.bbox then
2073       attr[#attr+1] = format("/BBox [%s]", opts.bbox)
2074     end
2075     local index = tex.saveboxresource(boxid, tableconcat(attr), optres, true, opts.bbox and 4 or 1)
```

```

2076 patterns[name] = { id = index, colored = opts.colored }
2077 else
2078   local objname = "@mplibpattern"..name
2079   local metric = format("bbox %s", opts.bbox or format("0 0 %s %s",wd,hd))
2080   local optres, t = opts.resources or "", { }
2081   if pdfmanagement then
2082     for _,v in ipairs{"ExtGState","ColorSpace","Shading"} do
2083       run_tex_code(format(
2084         [[\mplibtmptoks\expanded{{{/%s \pdf_object_ref:n{__pdf/Page/Resources/%s}}}],v,v),ccexplat)
2085       t[#t+1] = texgettoks'mplibtmptoks'
2086     end
2087   elseif is_defined(pdfetcs.pgfextgs) then
2088     run_tex_code("\mplibtmptoks\expanded{{\z
2089     \ifpgf@sys@pdf@extgs@exists /ExtGState @pgfextgs\fi\z
2090     \ifpgf@sys@pdf@colorspaces@exists /ColorSpace @pgfcolorspaces\fi}}",catat11)
2091     t[#t+1] = texgettoks'mplibtmptoks'
2092   else
2093     t[#t+1] = "/ExtGState @MPLibTr/Shading @MPLibSh/ColorSpace @MPLibCS"
2094   end
2095   optres = optres .. tableconcat(t)
2096   texsprint {
2097     [[\ifvmode\nointerlineskip\fi]],
2098     format([[ \hbox to\opt{\vbox to\opt{\vsize=\wd %i\vss\noindent}], boxid), -- force horiz mode?
2099     [[\special{pdf:bcontent}]],
2100     [[\special{pdf:boxobj }]], objname, format(" %s", metric),
2101     format([[ \raise\dp %i\box %i]], boxid, boxid),
2102     format([[ \special{pdf:put @resources <<%s>>}]], optres),
2103     [[\special{pdf:exobj <<]], tableconcat(attr), ">>"],
2104     [[\special{pdf:econtent}]],
2105     [[\par}\hss]],
2106   }
2107   patterns[#patterns+1] = objname
2108   patterns[name] = { id = #patterns, colored = opts.colored }
2109 end
2110 end
2111 local function pattern_colorspace (cs)
2112   local on, new = update_pdfobjs(format("/Pattern %s]", cs))
2113   if new then
2114     local key = format("MPLibCS%i",on)
2115     local val = pdfmode and format("%i 0 R",on) or format("@mplibpdfobj%i",on)
2116     if pdfmanagement then
2117       texsprint(ccexplat,format("\pdfmanagement_add:nnn{Page/Resources/ColorSpace}{%s}{%s}",key,val))
2118     else
2119       local res = format("/%s %s", key, val)
2120       if is_defined(pdfetcs.pgfcOLORSPACE) then
2121         texsprint(format("\csname %s\endcsname{%s}", pdfetcs.pgfcOLORSPACE, res))
2122       elseif pdfmode then
2123         if not pdfetcs.ColorSpace_res then
2124           pdfetcs.initialize_resources"ColorSpace"
2125         end
2126         pdfetcs.fallback_update_resources("ColorSpace", res)
2127       else
2128         texsprint(format("\special{pdf:put @MPLibCS<<%s>>}", res))
2129         texsprint("\special{pdf:put @resources<</ColorSpace @MPLibCS>>}")

```

```

2130     end
2131   end
2132 end
2133 return on
2134 end
2135 local function do_preobj_PAT(object, prescript)
2136   local name = prescript and prescript.mplibpattern
2137   if not name then return end
2138   local patt = patterns[name]
2139   local index = patt and patt.id or err("cannot get pattern object '%s'", name)
2140   local key = format("MPLibPt%s", index)
2141   if patt.colored then
2142     pdf_literalcode("/Pattern cs /%s scn", key)
2143   else
2144     local color = prescript.mpliboverridecolor
2145     if not color then
2146       local t = object.color
2147       color = t and #t>0 and luamplib.colorconverter(t)
2148     end
2149     if not color then return end
2150     local cs
2151     if color:find" cs " or color:find"@pdf.obj" then
2152       local t = color:explode()
2153       if pdfmode then
2154         cs = format("%s 0 R", ltx.pdf.object_id( t[1]:sub(2,-1) ))
2155         color = t[3]
2156       else
2157         cs = t[2]
2158         color = t[3]:match"%[(.+)%"
2159       end
2160     else
2161       local t = colorsplit(color)
2162       cs = #t == 4 and "/DeviceCMYK" or #t == 3 and "/DeviceRGB" or "/DeviceGray"
2163       color = tableconcat(t, " ")
2164     end
2165     pdf_literalcode("/MPLibCS%i cs %s /%s scn", pattern_colorspace(cs), color, key)
2166   end
2167   if patt.done then return end
2168   local val = pdfmode and format("%s 0 R", index) or patterns[index]
2169   if pdfmanagement then
2170     texsprint(ccexplat, format("\\pdfmanagement_add:nnn{Page/Resources/Pattern}{%s}{%s}", key, val))
2171   else
2172     local res = format("/%s %s", key, val)
2173     if is_defined(pdfetcs.pgfpattern) then
2174       texsprint(format("\\csname %s\\endcsname{%s}", pdfetcs.pgfpattern, res))
2175     elseif pdfmode then
2176       if not pdfetcs.Pattern_res then
2177         pdfetcs.initialize_resources"Pattern"
2178       end
2179       pdfetcs.fallback_update_resources("Pattern", res)
2180     else
2181       texsprint(format("\\special{pdf:put @MPLibPt<<%s>>}", res))
2182       texsprint("\\special{pdf:put @resources<</Pattern @MPLibPt>>}")
2183     end
2184   end

```

```

2184 end
2185 patt.done = true
2186 end
2187

```

Finally, flush figures by inserting PDF literals.

```

2188 function luamplib.flush (result, flusher)
2189   if result then
2190     local figures = result.fig
2191     if figures then
2192       for f=1, #figures do
2193         info("flushing figure %s", f)
2194         local figure = figures[f]
2195         local objects = getobjects(result, figure, f)
2196         local fignum = tonumber(figure:filename():match("[%d]+$") or figure:charcode() or 0)
2197         local miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
2198         local bbox = figure:boundingbox()
2199         local llx, lly, urx, ury = bbox[1], bbox[2], bbox[3], bbox[4] -- faster than unpack
2200         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()

```

```

2201     else

```

For legacy behavior, insert ‘pre-fig’ TeX code here.

```

2202     if tex_code_pre_mplib[f] then
2203       put2output(tex_code_pre_mplib[f])
2204     end
2205     pdf_startfigure(fignum, llx, lly, urx, ury)
2206     start_pdf_code()
2207     if objects then
2208       local savedpath = nil
2209       local savedhtap = nil
2210       for o=1, #objects do
2211         local object      = objects[o]
2212         local objecttype  = object.type

```

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

```

2213         local prescript      = object.prescript
2214         prescript = prescript and script2table(prescript) -- prescript is now a table
2215         local cr_over = do_preobj_CR(object, prescript) -- color
2216         local tr_opaq = do_preobj_TR(object, prescript) -- opacity
2217         if prescript and prescript.mplibtexboxid then
2218           put_tex_boxes(object, prescript)
2219         elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then --skip
2220         elseif objecttype == "start_clip" then
2221           local evenodd = not object.istext and object.postscript == "evenodd"
2222           start_pdf_code()
2223           flushnormalpath(object.path, false)

```

```

2224         pdf_literalcode(evenodd and "W* n" or "W n")
2225     elseif objecttype == "stop_clip" then
2226         stop_pdf_code()
2227         miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
2228     elseif objecttype == "special" then

```

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

```

2229         if prescript and prescript.postmplibverbtx then
2230             figcontents.post[#figcontents.post+1] = prescript.postmplibverbtx
2231         end
2232     elseif objecttype == "text" then
2233         local ot = object.transform -- 3,4,5,6,1,2
2234         start_pdf_code()
2235         pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
2236         pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
2237         stop_pdf_code()
2238     else
2239         local evenodd, collect, both = false, false, false
2240         local postscript = object.postscript
2241         if not object.istext then
2242             if postscript == "evenodd" then
2243                 evenodd = true
2244             elseif postscript == "collect" then
2245                 collect = true
2246             elseif postscript == "both" then
2247                 both = true
2248             elseif postscript == "eoboth" then
2249                 evenodd = true
2250                 both = true
2251             end
2252         end
2253         if collect then
2254             if not savedpath then
2255                 savedpath = { object.path or false }
2256                 savedhtap = { object.htap or false }
2257             else
2258                 savedpath[#savedpath+1] = object.path or false
2259                 savedhtap[#savedhtap+1] = object.htap or false
2260             end
2261         else

```

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

```

2262         local shade_no = do_preobj_SH(object,prescript) -- shading
2263         local pattern_ = do_preobj_PAT(object,prescript) -- pattern
2264         local ml = object.miterlimit
2265         if ml and ml ~= miterlimit then
2266             miterlimit = ml
2267             pdf_literalcode("%f M",ml)
2268         end
2269         local lj = object.linejoin
2270         if lj and lj ~= linejoin then
2271             linejoin = lj
2272             pdf_literalcode("%i j",lj)
2273         end
2274         local lc = object.linecap

```

```

2275     if lc and lc ~= linecap then
2276         linecap = lc
2277         pdf_literalcode("%i J",lc)
2278     end
2279     local dl = object.dash
2280     if dl then
2281         local d = format("[%s] %f d",tableconcat(dl.dashes or {}," "),dl.offset)
2282         if d ~= dashed then
2283             dashed = d
2284             pdf_literalcode(dashed)
2285         end
2286     elseif dashed then
2287         pdf_literalcode("[[] 0 d")
2288         dashed = false
2289     end
2290     local path = object.path
2291     local transformed, penwidth = false, 1
2292     local open = path and path[1].left_type and path[#path].right_type
2293     local pen = object.pen
2294     if pen then
2295         if pen.type == 'elliptical' then
2296             transformed, penwidth = pen_characteristics(object) -- boolean, value
2297             pdf_literalcode("%f w",penwidth)
2298             if objecttype == 'fill' then
2299                 objecttype = 'both'
2300             end
2301         else -- calculated by mplib itself
2302             objecttype = 'fill'
2303         end
2304     end
2305     if transformed then
2306         start_pdf_code()
2307     end
2308     if path then
2309         if savedpath then
2310             for i=1,#savedpath do
2311                 local path = savedpath[i]
2312                 if transformed then
2313                     flushconcatpath(path,open)
2314                 else
2315                     flushnormalpath(path,open)
2316                 end
2317             end
2318             savedpath = nil
2319         end
2320         if transformed then
2321             flushconcatpath(path,open)
2322         else
2323             flushnormalpath(path,open)
2324         end
2325     end
2326     if not shade_no then -- conflict with shading
2327         if objecttype == "fill" then
2328             pdf_literalcode(evenodd and "h f*" or "h f")

```

```

2328         elseif objecttype == "outline" then
2329             if both then
2330                 pdf_literalcode(evenodd and "h B*" or "h B")
2331             else
2332                 pdf_literalcode(open and "S" or "h S")
2333             end
2334         elseif objecttype == "both" then
2335             pdf_literalcode(evenodd and "h B*" or "h B")
2336         end
2337     end
2338 end
2339 if transformed then
2340     stop_pdf_code()
2341 end
2342 local path = object.htap
2343 if path then
2344     if transformed then
2345         start_pdf_code()
2346     end
2347     if savedhtap then
2348         for i=1,#savedhtap do
2349             local path = savedhtap[i]
2350             if transformed then
2351                 flushconcatpath(path,open)
2352             else
2353                 flushnormalpath(path,open)
2354             end
2355         end
2356         savedhtap = nil
2357         evenodd = true
2358     end
2359     if transformed then
2360         flushconcatpath(path,open)
2361     else
2362         flushnormalpath(path,open)
2363     end
2364     if objecttype == "fill" then
2365         pdf_literalcode(evenodd and "h f*" or "h f")
2366     elseif objecttype == "outline" then
2367         pdf_literalcode(open and "S" or "h S")
2368     elseif objecttype == "both" then
2369         pdf_literalcode(evenodd and "h B*" or "h B")
2370     end
2371     if transformed then
2372         stop_pdf_code()
2373     end
2374 end

```

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

```

2375         if shade_no then -- shading
2376             pdf_literalcode("W n /MPLibSh%s sh Q",shade_no)
2377         end
2378     end
2379 end
2380 if tr_opaq then -- opacity

```

```

2381         stop_pdf_code()
2382     end
2383     if cr_over then -- color
2384         put2output"\special{pdf:ec}"
2385     end
2386 end
2387 end
2388 stop_pdf_code()
2389 pdf_stopfigure()

```

output collected materials to PDF, plus legacy verbatimex code.

```

2390     for _,v in ipairs(figcontents) do
2391         if type(v) == "table" then
2392             texsprint"\mplibtoPDF{"; texsprint(v[1], v[2]); texsprint"}"
2393         else
2394             texsprint(v)
2395         end
2396     end
2397     if #figcontents.post > 0 then texsprint(figcontents.post) end
2398     figcontents = { post = { } }
2399 end
2400 end
2401 end
2402 end
2403 end
2404
2405 function luamplib.colorconverter (cr)
2406     local n = #cr
2407     if n == 4 then
2408         local c, m, y, k = cr[1], cr[2], cr[3], cr[4]
2409         return format("%.3f %.3f %.3f %.3f k %.3f %.3f %.3f %.3f K",c,m,y,k,c,m,y,k), "0 g 0 G"
2410     elseif n == 3 then
2411         local r, g, b = cr[1], cr[2], cr[3]
2412         return format("%.3f %.3f %.3f rg %.3f %.3f %.3f RG",r,g,b,r,g,b), "0 g 0 G"
2413     else
2414         local s = cr[1]
2415         return format("%.3f g %.3f G",s,s), "0 g 0 G"
2416     end
2417 end

```

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

First we need to load some packages.

```

2418 \bgroup\expandafter\expandafter\expandafter\egroup
2419 \expandafter\ifx\csname selectfont\endcsname\relax
2420 \input ltluatex
2421 \else
2422 \NeedsTeXFormat{LaTeX2e}
2423 \ProvidesPackage{luamplib}
2424 [2024/06/13 v2.32.1 mplib package for LuaTeX]
2425 \ifx\newluafunction\@undefined
2426 \input ltluatex
2427 \fi
2428 \fi

```

Loading of lua code.

```
2429 \directlua{require("luamplib")}
```

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

```
2430 \ifx\pdfoutput\undefined
```

```
2431 \let\pdfoutput\outputmode
```

```
2432 \fi
```

```
2433 \ifx\pdfliteral\undefined
```

```
2434 \protected\def\pdfliteral{\pdfextension literal}
```

```
2435 \fi
```

Set the format for metapost.

```
2436 \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.

```
2437 \ifnum\pdfoutput>0
```

```
2438 \let\mplibtoPDF\pdfliteral
```

```
2439 \else
```

```
2440 \def\mplibtoPDF#1{\special{pdf:literal direct #1}}
```

```
2441 \ifcsname PackageInfo\endcsname
```

```
2442 \PackageInfo{luamplib}{only dvipdfmx is supported currently}
```

```
2443 \else
```

```
2444 \immediate\write-1{luamplib Info: only dvipdfmx is supported currently}
```

```
2445 \fi
```

```
2446 \fi
```

To make mplibcode typeset always in horizontal mode.

```
2447 \def\mplibforcehmode{\let\prependtomplibbox\leavevmode}
```

```
2448 \def\mplibnoforcehmode{\let\prependtomplibbox\relax}
```

```
2449 \mplibnoforcehmode
```

Catcode. We want to allow comment sign in mplibcode.

```
2450 \def\mplibsetupcatcodes{%
```

```
2451 %catcode`\{=12 %catcode`\}=12
```

```
2452 \catcode`\#=12 \catcode`\^=12 \catcode`\~=12 \catcode`\_=12
```

```
2453 \catcode`\&=12 \catcode`\$=12 \catcode`\%=12 \catcode`\^M=12
```

```
2454 }
```

Make btex...etex box zero-metric.

```
2455 \def\mplibputtextbox#1{\vbox to 0pt{\vss\hbox to 0pt{\raise\dp#1\copy#1\hss}}}
```

Patterns

```
2456 {\def\:\global\let\mplibsptoken= } \: }
```

```
2457 \protected\def\mplibpattern#1{%
```

```
2458 \begingroup
```

```
2459 \def\mplibpatternname{#1}%
```

```
2460 \mplibpatterngetnexttok
```

```
2461 }
```

```
2462 \def\mplibpatterngetnexttok{\futurelet\nexttok\mplibpatternbranch}
```

```
2463 \def\mplibpatternskipspace{\afterassignment\mplibpatterngetnexttok\let\nexttok= }
```

```
2464 \def\mplibpatternbranch{%
```

```
2465 \ifx [\nexttok
```

```
2466 \expandafter\mplibpatternopts
```

```
2467 \else
```

```
2468 \ifx\mplibsptoken\nexttok
```

```

2469     \expandafter\expandafter\expandafter\mplibpatternskip space
2470     \else
2471       \let\mplibpatternoptions\empty
2472       \expandafter\expandafter\expandafter\mplibpatternmain
2473     \fi
2474 \fi
2475 }
2476 \def\mplibpatternopts[#1]{%
2477 \def\mplibpatternoptions{#1}%
2478 \mplibpatternmain
2479 }
2480 \def\mplibpatternmain{%
2481 \setbox\mplibscratchbox\hbox\bgroup\ignorespaces
2482 }
2483 \protected\def\endmppattern{%
2484 \egroup
2485 \directlua{ luamplib.registerpattern(
2486 \the\mplibscratchbox, '\mplibpatternname', {\mplibpatternoptions}
2487 )}%
2488 \endgroup
2489 }

    simple way to use mplib: \mpfig draw fullcircle scaled 10; \endmpfig
2490 \def\mpfiginstancename{@mpfig}
2491 \protected\def\mpfig{%
2492 \begingroup
2493 \futurelet\nexttok\mplibmpfigbranch
2494 }
2495 \def\mplibmpfigbranch{%
2496 \ifx *\nexttok
2497 \expandafter\mplibprempfig
2498 \else
2499 \expandafter\mplibmainmpfig
2500 \fi
2501 }
2502 \def\mplibmainmpfig{%
2503 \begingroup
2504 \mplibsetupcatcodes
2505 \mplibdomainmpfig
2506 }
2507 \long\def\mplibdomainmpfig#1\endmpfig{%
2508 \endgroup
2509 \directlua{
2510 local legacy = luamplib.legacy_verbatimtex
2511 local everympfig = luamplib.everymplib["\mpfiginstancename"] or ""
2512 local everyendmpfig = luamplib.everyendmplib["\mpfiginstancename"] or ""
2513 luamplib.legacy_verbatimtex = false
2514 luamplib.everymplib["\mpfiginstancename"] = ""
2515 luamplib.everyendmplib["\mpfiginstancename"] = ""
2516 luamplib.process_mplibcode(
2517 "beginfig(0) ".everympfig.." "..[===[\unexpanded{#1}]===].." ".everyendmpfig.." endfig;",
2518 "\mpfiginstancename")
2519 luamplib.legacy_verbatimtex = legacy
2520 luamplib.everymplib["\mpfiginstancename"] = everympfig
2521 luamplib.everyendmplib["\mpfiginstancename"] = everyendmpfig

```

```

2522 }%
2523 \endgroup
2524 }
2525 \def\mplibprempfig#1{%
2526 \begingroup
2527 \mplibsetupcatcodes
2528 \mplibdoprempfig
2529 }
2530 \long\def\mplibdoprempfig#1\endmpfig{%
2531 \endgroup
2532 \directlua{
2533   local legacy = luamplib.legacy_verbatimex
2534   local everympfig = luamplib.everymplib["\mpfiginstancename"]
2535   local everyendmpfig = luamplib.everyendmplib["\mpfiginstancename"]
2536   luamplib.legacy_verbatimex = false
2537   luamplib.everymplib["\mpfiginstancename"] = ""
2538   luamplib.everyendmplib["\mpfiginstancename"] = ""
2539   luamplib.process_mplibcode(===[\unexpanded{#1}]===, "\mpfiginstancename")
2540   luamplib.legacy_verbatimex = legacy
2541   luamplib.everymplib["\mpfiginstancename"] = everympfig
2542   luamplib.everyendmplib["\mpfiginstancename"] = everyendmpfig
2543 }%
2544 \endgroup
2545 }
2546 \protected\def\endmpfig{endmpfig}

```

The Plain-specific stuff.

```

2547 \unless\ifcsname ver@luamplib.sty\endcsname
2548 \def\mplibcodegetinstancename[#1]{\gdef\currentmpinstancename{#1}\mplibcodeindeed}
2549 \protected\def\mplibcode{%
2550 \begingroup
2551 \futurelet\nexttok\mplibcodebranch
2552 }
2553 \def\mplibcodebranch{%
2554 \ifx [\nexttok
2555 \expandafter\mplibcodegetinstancename
2556 \else
2557 \global\let\currentmpinstancename\empty
2558 \expandafter\mplibcodeindeed
2559 \fi
2560 }
2561 \def\mplibcodeindeed{%
2562 \begingroup
2563 \mplibsetupcatcodes
2564 \mplibdocode
2565 }
2566 \long\def\mplibdocode#1\endmplibcode{%
2567 \endgroup
2568 \directlua{luamplib.process_mplibcode(===[\unexpanded{#1}]===, "\currentmpinstancename")}%
2569 \endgroup
2570 }
2571 \protected\def\endmplibcode{endmplibcode}
2572 \else

```

The  $\LaTeX$ -specific part: a new environment.

```

2573 \newenvironment{mplibcode}[1][{}]{%
2574   \global\def\currentmpinstancename{#1}%
2575   \mplibmptoks{}\ltxdomplibcode
2576 }{}
2577 \def\ltxdomplibcode{%
2578   \begingroup
2579   \mplibsetupcatcodes
2580   \ltxdomplibcodeindeed
2581 }
2582 \def\mplib@mplibcode{mplibcode}
2583 \long\def\ltxdomplibcodeindeed#1\end#2{%
2584   \endgroup
2585   \mplibmptoks\expandafter{\the\mplibmptoks#1}%
2586   \def\mplibtemp@a{#2}%
2587   \ifx\mplib@mplibcode\mplibtemp@a
2588     \directlua{luamplib.process_mplibcode([===[\the\mplibmptoks]===],"\currentmpinstancename")}%
2589     \end{mplibcode}%
2590   \else
2591     \mplibmptoks\expandafter{\the\mplibmptoks\end{#2}}%
2592     \expandafter\ltxdomplibcode
2593   \fi
2594 }
2595 \fi

```

#### User settings.

```

2596 \def\mplibshowlog#1{\directlua{
2597   local s = string.lower("#1")
2598   if s == "enable" or s == "true" or s == "yes" then
2599     luamplib.showlog = true
2600   else
2601     luamplib.showlog = false
2602   end
2603 }}
2604 \def\mpliblegacybehavior#1{\directlua{
2605   local s = string.lower("#1")
2606   if s == "enable" or s == "true" or s == "yes" then
2607     luamplib.legacy_verbatimtex = true
2608   else
2609     luamplib.legacy_verbatimtex = false
2610   end
2611 }}
2612 \def\mplibverbatim#1{\directlua{
2613   local s = string.lower("#1")
2614   if s == "enable" or s == "true" or s == "yes" then
2615     luamplib.verbatiminput = true
2616   else
2617     luamplib.verbatiminput = false
2618   end
2619 }}
2620 \newtoks\mplibmptoks

```

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

```

2621 \ifcsname ver@luamplib.sty\endcsname
2622 \protected\def\everymplib{%
2623   \begingroup

```

```

2624 \mplibsetupcatcodes
2625 \mplibdoeverymplib
2626 }
2627 \protected\def\everyendmplib{%
2628 \begingroup
2629 \mplibsetupcatcodes
2630 \mplibdoeveryendmplib
2631 }
2632 \newcommand\mplibdoeverymplib[2][]{%
2633 \endgroup
2634 \directlua{
2635   luamplib.everymplib["#1"] = [===[\unexpanded{#2}]===]
2636 }%
2637 }
2638 \newcommand\mplibdoeveryendmplib[2][]{%
2639 \endgroup
2640 \directlua{
2641   luamplib.everyendmplib["#1"] = [===[\unexpanded{#2}]===]
2642 }%
2643 }
2644 \else
2645 \def\mplibgetinstancename[#1]{\def\currentmpinstancename{#1}}
2646 \protected\def\everymplib#1#1{%
2647 \ifx\empty#1\empty \mplibgetinstancename[]\else \mplibgetinstancename#1\fi
2648 \begingroup
2649 \mplibsetupcatcodes
2650 \mplibdoeverymplib
2651 }
2652 \long\def\mplibdoeverymplib#1{%
2653 \endgroup
2654 \directlua{
2655   luamplib.everymplib["\currentmpinstancename"] = [===[\unexpanded{#1}]===]
2656 }%
2657 }
2658 \protected\def\everyendmplib#1#1{%
2659 \ifx\empty#1\empty \mplibgetinstancename[]\else \mplibgetinstancename#1\fi
2660 \begingroup
2661 \mplibsetupcatcodes
2662 \mplibdoeveryendmplib
2663 }
2664 \long\def\mplibdoeveryendmplib#1{%
2665 \endgroup
2666 \directlua{
2667   luamplib.everyendmplib["\currentmpinstancename"] = [===[\unexpanded{#1}]===]
2668 }%
2669 }
2670 \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.

```

2671 \def\mpdim#1{ runscript("luamplibdimen{#1}") }
2672 \def\mpcolor#1#1{\domplibcolor{#1}}
2673 \def\domplibcolor#1#2{ runscript("luamplibcolor{#1}{#2}") }

```

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

```
2674 \def\mplibnumbersystem#1{\directlua{
2675   local t = "#1"
2676   if t == "binary" then t = "decimal" end
2677   luamplib.numbersystem = t
2678 }}
```

Settings for .mp cache files.

```
2679 \def\mplibmakenocache#1{\mplibdomakenocache #1,*}
2680 \def\mplibdomakenocache#1,{%
2681   \ifx\empty#1\empty
2682   \expandafter\mplibdomakenocache
2683   \else
2684   \ifx*#1\else
2685     \directlua{luamplib.noneedtoreplace["#1.mp"]=true}%
2686     \expandafter\expandafter\expandafter\mplibdomakenocache
2687   \fi
2688   \fi
2689 }
2690 \def\mplibcancelnocache#1{\mplibdocancelnocache #1,*}
2691 \def\mplibdocancelnocache#1,{%
2692   \ifx\empty#1\empty
2693   \expandafter\mplibdocancelnocache
2694   \else
2695   \ifx*#1\else
2696     \directlua{luamplib.noneedtoreplace["#1.mp"]=false}%
2697     \expandafter\expandafter\expandafter\mplibdocancelnocache
2698   \fi
2699   \fi
2700 }
2701 \def\mplibcachedir#1{\directlua{luamplib.getcachedir("\unexpanded{#1}")}}
```

More user settings.

```
2702 \def\mplibtexttextlabel#1{\directlua{
2703   local s = string.lower("#1")
2704   if s == "enable" or s == "true" or s == "yes" then
2705     luamplib.texttextlabel = true
2706   else
2707     luamplib.texttextlabel = false
2708   end
2709 }}
2710 \def\mplibcodeinherit#1{\directlua{
2711   local s = string.lower("#1")
2712   if s == "enable" or s == "true" or s == "yes" then
2713     luamplib.codeinherit = true
2714   else
2715     luamplib.codeinherit = false
2716   end
2717 }}
2718 \def\mplibglobaltexttext#1{\directlua{
2719   local s = string.lower("#1")
2720   if s == "enable" or s == "true" or s == "yes" then
2721     luamplib.globaltexttext = true
2722   else
2723     luamplib.globaltexttext = false
```

```
2724 end
2725 }}
```

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

```
2726 \ifx\mplibscratchbox\undefined \newbox\mplibscratchbox \fi
```

We encapsulate the literals.

```
2727 \def\mplibstarttoPDF#1#2#3#4{%
2728   \prependtomplibbox
2729   \hbox dir TLT\bgroup
2730   \xdef\MPllx{#1}\xdef\MPlly{#2}%
2731   \xdef\MPurx{#3}\xdef\MPury{#4}%
2732   \xdef\MPwidth{\the\dimexpr#3bp-#1bp\relax}%
2733   \xdef\MPheight{\the\dimexpr#4bp-#2bp\relax}%
2734   \parskip0pt%
2735   \leftskip0pt%
2736   \parindent0pt%
2737   \everypar{}%
2738   \setbox\mplibscratchbox\vbox\bgroup
2739   \noindent
2740 }
2741 \def\mplibstoptoPDF{%
2742   \par
2743   \egroup %
2744   \setbox\mplibscratchbox\hbox %
2745     {\hskip-\MPllx bp%
2746      \raise-\MPlly bp%
2747      \box\mplibscratchbox}%
2748   \setbox\mplibscratchbox\vbox to \MPheight
2749     {\vfill
2750      \hsize\MPwidth
2751      \wd\mplibscratchbox0pt%
2752      \ht\mplibscratchbox0pt%
2753      \dp\mplibscratchbox0pt%
2754      \box\mplibscratchbox}%
2755   \wd\mplibscratchbox\MPwidth
2756   \ht\mplibscratchbox\MPheight
2757   \box\mplibscratchbox
2758   \egroup
2759 }
```

Text items have a special handler.

```
2760 \def\mplibtexttext#1#2#3#4#5{%
2761   \begingroup
2762   \setbox\mplibscratchbox\hbox
2763     {\font\temp=#1 at #2bp%
2764      \temp
2765      #3}%
2766   \setbox\mplibscratchbox\hbox
2767     {\hskip#4 bp%
2768      \raise#5 bp%
2769      \box\mplibscratchbox}%
2770   \wd\mplibscratchbox0pt%
2771   \ht\mplibscratchbox0pt%
2772   \dp\mplibscratchbox0pt%
```

```
2773 \box\mplibscratchbox
2774 \endgroup
2775 }
    Input luamplib.cfg when it exists.
2776 \openin0=luamplib.cfg
2777 \ifeof0 \else
2778 \closein0
2779 \input luamplib.cfg
2780 \fi
    That's all folks!
```

# 3 The GNU GPL License v2

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

## GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

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

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

### Preamble

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

### NO WARRANTY

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

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

### END OF TERMS AND CONDITIONS

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

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

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

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

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

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

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

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

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

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

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

Vorodyne, Inc., hereby disclaims all copyright interest in the program "GNUconviption" (which makes passes at compilers) written by James Hacker.

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

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.