The luamplib package

Hans Hagen, Taco Hoekwater, Elie Roux, Philipp Gesang and Kim Dohyun
Current Maintainer: Kim Dohyun
Support: https://github.com/lualatex/luamplib

2024/07/17 v2.34.0

Abstract

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

1 Documentation

This package 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 for the Lua **mplib** functions and some TEX functions to have the output of the **mplib** functions in the pdf.

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

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

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

- possibility to use `btex ... etex` to typeset TEX code. `textext()` is a more versatile macro equivalent to `TEX()` from TEX.**mp**. `TEX()` is also allowed and is a synonym of `textext()`. The argument of **mplib**'s primitive `maketext` will also be processed by the same routine.
- possibility to use `verbatimtex ... etex`, though it's behavior cannot be the same as the stand-alone **mpost**. Of course you cannot include `\documentclass`, `\usepackage` etc. When these TEX commands are found in `verbatimtex ... etex`, the entire code will be ignored. The treatment of `verbatimtex` command has changed a lot since v2.20; see below regarding `\mpliblegacybehavior`.
- in the past, the package required PDF mode in order to have some output. Starting with version 2.7 it works in DVI mode as well, though DVIPDFMx is the only DVI tool currently supported.

It seems to be convenient to divide the explanations of some more changes and cautions into three parts: TEX, MetaPost, and Lua interfaces.

1.1 TEX

\mplibforcehmode When this macro is declared, every metapost 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 redefine this command with anything suitable before a box.)

\everymplib{...}, \everyendmplib{...} \everymplib and \everyendmplib redefine the lua table containing metapost code which will be automatically inserted at the beginning and ending of each metapost code chunk.

```
\everymplib{ beginfig(0); }
\everyendmplib{ endfig; }
\begin{mplibcode}
  % beginfig/endfig not needed
  draw fullcircle scaled 1cm;
\end{mplibcode}
```

\mplibsetformat{plain|metafun} 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*>}.

N.B. As *metafun* is such a complicated format, we cannot support all the functionalities producing special effects provided by *metafun*. At least, however, transparency (actually opacity), transparency group, and shading (gradient colors) are fully supported, and outlinetext is supported by our own alternative mpliboutlinetext (see below § 1.2).

☞ Among these, transparency is so simple that you can apply it to an object, even with the *plain* format, just by appending withprescript "tr_transparency=<*number*>" to the sentence. ($0 \leq$ <*number*> ≤ 1)

As for transparency group, the current *metafun* document § 8.8 is not correct. The true syntax is:

```
draw <picture>|<path> asgroup <string>
```

where <*string*> should be "" (empty), "isolated", "knockout", or "isolated,knockout". Beware that currently many of the PDF rendering applications, except Adobe Acrobat Reader, cannot properly render the isolated or knockout effect.

One thing worth mentioning about shading is: when a color expression is given in string type, it is regarded by luamplib as a color expression of TEX side. For instance, when withshadecolors("orange", 2/3red) is given, the first color "orange" will be interpreted as an xcolor's or l3color's expression.

\mplibnumbersystem{scaled|double|decimal} Users can choose numbersystem option. The default value is scaled, which can be changed by declaring \mplibnumbersystem{double} or \mplibnumbersystem{decimal}.

\mplibshowlog{enable|disable} Default: disable. When \mplibshowlog{enable}[1] is declared, log messages returned by the metapost process will be printed to the .log file. This is the TEX side interface for luamplib.showlog.

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

\mpliblegacybehavior{enable|disable} By default, \mpliblegacybehavior{enable} is already declared for backward compatibility, in which case TEX code in verbatimtex ... etex that comes just before beginfig() will be inserted before the following metapost figure box. In this way, each figure box can be freely moved horizontally or vertically. Also, a box number can be assigned to a figure box, allowing it to be reused later.

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

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

On the other hand, TEX code in verbatimtex ... etex between beginfig() and endfig will be inserted after flushing out the metapost figure. As shown in the example below, VerbatimTeX() is a synonym of verbatimtex ... etex.

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

By contrast, when \mpliblegacybehavior{disabled} is declared, 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 following btex ... etex codes.

```
\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}
```

\mplibtextextlabel{enable|disable} Default: disable. \mplibtextextlabel{enable} enables the labels typeset via textext instead of infont operator. So, label("my text",origin) thereafter is exactly the same as label(textext("my text"),origin).

N.B. In the background, luamplib redefines infont operator so that the right side argument (the font part) is totally ignored. Therefore the left side arguemnt will be typeset with the current TEX font. Also take care of char operator in the left side argument, as this might bring unpermitted characters into TEX.

\mplibcodeinherit{enable|disable} Default: disable. \mplibcodeinherit{enable} enables the inheritance of variables, constants, and macros defined by previous metapost code chunks. On the contrary, \mplibcodeinherit{disable} will make each code chunk being treated as an independent instance, never affected by previous code chunks.

Separate MetaPost instances luamplib v2.22 has added the support for several named metapost instances in LaTeX `mplibcode` environment. Plain TeX users also can use this functionality. The syntax for LaTeX is:

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

The behavior 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).
- `btex ... etex` boxes are also shared and do not require `\mplibglobaltextext`.
- When an instance names is set, respective `\currentmpinstancename` is set as well.

In parellel with this functionality, we support 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`. The syntax is:

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

`\mplibglobaltextext{enable|disable}` Default: `disable`. Formerly, to inherit `btex ... etex` boxes as well as other metapost macros, variables and constants, it was necessary to declare `\mplibglobaltextext{enable}` in advance. But from v2.27, this is implicitly enabled when `\mplibcodeinherit` is enabled. This optional command still remains mostly for backward compatibility.

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

`\mplibverbatim{enable|disable}` Default: `disable`. Users can issue `\mplibverbatim{enable}`, after which the contents of mplibcode environment will be read verbatim. As a result, except for `\mpdim` and `\mpcolor` (see below), all other TeX commands outside of the `btex` or `verbatimtex ... etex` are not expanded and will be fed literally to the **mplib** library.

\mpdim{...} Besides other TEX commands, \mpdim is specially allowed in the mplibcode environment. This feature is inpired by **gmp** package authored by Enrico Gregorio. Please refer to the manual of **gmp** package for details.

```
\begin{mplibcode}
  beginfig(1)
  draw origin--(.6\mpdim{\linewidth},0) withpen pencircle scaled 4
  dashed evenly scaled 4 withcolor \mpcolor{orange};
  endfig;
\end{mplibcode}
```

\mpcolor[...]{...} With \mpcolor command, color names or expressions of **color**, **xcolor** and **l3color** module/packages can be used in the mplibcode enviroment (after withcolor operator). See the example above. The optional [...] means the option of **xcolor**'s \color command. For spot colors, **l3color** (in PDF/DVI mode), **colorspace**, **spotcolor** (in PDF mode) and **xespotcolor** (in DVI mode) packages are supported as well.

\mpfig ... \endmpfig Besides the mplibcode environment (for LaTeX) and \mplibcode ... \endmplibcode (for Plain), we also provide unexpandable TEX macros \mpfig ... \endmpfig and its starred version \mpfig* ... \endmpfig to save typing toil. The former 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} is forcibly declared. Again, as both share the same instance name, metapost codes are inherited among them. A simple example:

```
\everymplib[@mpfig]{ drawoptions(withcolor .5[red,white]); }
\mpfig* input boxes \endmpfig
\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} is not declared.

About cache files To support btex ... etex in external .mp files, luamplib inspects the content of each and every .mp file and makes caches if neceeccsary, before returning their paths to LuaTEX's **mplib** library. This could waste the compilation time, as most .mp files do not contain btex ... etex commands. So luamplib provides macros as follows, so that users can give instructions about files that do not require this functionality.

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

where <filename> is a filename 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.

Users can change this behavior by the command \mplibcachedir{<directory path>}, where tilde (~) is interpreted as the user's home directory (on a windows machine as well). As backslashes (\) should be escaped by users, it would be easier to use slashes (/) instead.

About figure box metric Notice that, after each figure is processed, the macro \MPwidth stores the width value of the latest figure; \MPheight, the height value. Incidentally, also note that \MPllx, \MPlly, \MPurx, and \MPury store the bounding box information of the 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.

1.2 MetaPost

mplibdimen(...), mplibcolor(...) These are MetaPost interfaces for the TEX commands \mpdim and \mpcolor. For example, mplibdimen("\linewidth") is basically the same as \mpdim{\linewidth}, and mplibcolor("red!50") is basically the same as \mpcolor{red!50}. The difference is that these metapost operators can also be used in external .mp files, which cannot have TEX commands outside of the btex or verbatimtex ... etex.

mplibtexcolor ..., mplibrgbtexcolor ... mplibtexcolor, which accepts a string argument, is a metapost operator that converts a TEX color expression to a MetaPost color expression, that can be used anywhere color expression is expected as well as after the withcolor operator. For instance:

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

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

mplibgraphictext ... mplibgraphictext is a metapost operator, the effect of which is similar to that of ConTEXt's graphictext or our own mpliboutlinetext (see below). However the syntax is somewhat different.

```
mplibgraphictext "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 the color expressions are given in string type, they are regarded as xcolor's or l3color's expressions. All from mplibgraphictext to the end of sentence will compose an anonymous picture, which can be drawn or assigned to a variable. Incidentally, withdrawcolor and withfillcolor are synonyms of drawcolor and fillcolor, hopefully to be compatible with graphictext.

N.B. In some cases, mplibgraphictext will produce better results than ConTEXt or even than our own mpliboutlinetext, especially when processing complicated TEX code such as the vertical writing in Chinese or Japanese. However, because the implementation is quite different from others, there are some limitations such that you can't apply shading (gradient colors) to the text. Again, in DVI mode, unicode-math package is needed for math formula, as we cannot embolden type1 fonts in DVI mode.

mplibglyph ... of ... From v2.30, we provide a new metapost operator mplibglyph, which returns a metapost picture containing outline paths of a glyph in opentype, truetype 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, repectively. 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.

mplibdrawglyph ... The picture returned by mplibglyph 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 the background color. In contrast, mplibdrawglyph *<picture>* 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.

☞ To apply the nonzero winding number rule to a picture containg paths, luamplib appends withpostscript "collect" to the paths except the last one in the picture. If you want the even-odd rule instead, you can, even with *plain* format, additionally declare withpostscript "evenodd" to the last path in the picture.

mpliboutlinetext (...) From v2.31, a new metapost operator mpliboutlinetext is available, 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, 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 ... TEX macros \mppattern{<name>} ... \endmppattern define 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> by replicating it horizontally and vertically. An example:

```
\mppattern{mypatt}          % or \begin{mppattern}{mypatt}
  [                         % options: see below
    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 listed in Table 1.

For the sake of convenience, the 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.

Table 1: options for \mppattern

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

* in string type, numbers are separated by spaces

Option colored=false (coloured is a synonym of colored) 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 = "slanted .3 rotated 30",
  ]
  \tiny\TeX
\end{mppattern}

\begin{mplibcode}
  beginfig(1)
  picture tex;
  tex = mpliboutlinetext.p ("\bfseries \TeX");
  for i=1 upto mpliboutlinenum:
    j:=0;
    for item within mpliboutlinepic[i]:
      j:=j+1;
      draw pathpart item scaled 10
      if j < length mpliboutlinepic[i]:
          withpostscript "collect"
      else:
          withpattern "pattuncolored"
          withpen pencircle scaled 1/2
          withcolor (i/4)[red,blue]          % paints the pattern
      fi;
    endfor
  endfor
  endfig;
\end{mplibcode}
```

... withfademethod ..., and related macros withfademethod is a metapost operator which makes the color of an object gradiently transparent. The syntax is <*path*>|<*picture*> withfademethod <*string*>, the latter being either "linear" or "circular". Though it is similar to the withshademethod provided by metafun, the differences are: (1) the operand of withfademethod can be a picture as well as a path; (2) you cannot make gradient colors, but can only make gradient opacity.

Related macros to control optional values are:

`withfadeopacity` (*number, number*) sets the starting opacity and the ending opacity, default value being `(1,0)`. '`1`' denotes full color; '`0`' full transparency.

`withfadevector` (*pair, pair*) sets the starting and ending points. Default value in the linear mode is `(llcorner p, lrcorner p)`, where `p` is the operand, meaning that fading starts from the left edge and ends at the right edge. Default value in the circular mode is `(center p, center p)`, which means centers of both starting and ending circles are the center of the bounding box.

`withfadecenter` is a synonym of `withfadevector`.

`withfaderadius` (*number, number*) sets the radii of starting and ending circles. This is no-op in the linear mode. Default value is `(0, abs(center p - urcorner p))`, meaning that fading starts from the center and ends at the four corners of the bounding box.

`withfadebbox` (*pair, pair*) sets the bounding box of the fading area, default value being `(llcorner p, urcorner p)`. Though this option is not needed in most cases, there could be cases when users want to explicitly control the bounding box.

An example:

```
\mpfig
  picture mill;
  mill = btex \includegraphics[width=100bp]{mill} etex;
  draw mill
    withfademethod  "circular"
    withfadecenter  (center mill, center mill)
    withfaderadius  (20, 50)
    withfadeopacity (1, 0)
    ;
\endmpfig
```

1.3 Lua

`runscript` ... Using the primitive `runscript` <*string*>, you can run a Lua code chunk from MetaPost side and get some metapost code returned by Lua if you want. As the functionality is provided by the **mplib** library itself, luamplib does not have much to say about it.

One thing is worth mentioning, however: if you return a Lua *table* to the metapost process, it is automatically converted to a relevant metapost value type such as pair, color, cmykcolor or transform. So users can save some extra toil of converting a table to a string, though it's not a big deal. For instance, `runscript "return {1,0,0}"` will give you the metapost color expression `(1,0,0)` automatically.

Lua table `luamplib.instances` Users can access the Lua table containing mplib instances, `luamplib.instances`, through which metapost variables are also easily accessible from Lua side, as documented in LuaTeX manual § 11.2.8.4 (`texdoc luatex`). The following will print `false`, `3.0`, `MetaPost` and the knots and the cyclicity of the path `unitsquare`, consecutively.

```
\begin{mplibcode}[instance1]
```

Table 2: elements in luamplib table (partial)

Key	Type	Related TEX macro
codeinherit	*boolean*	\mplibcodeinherit
everyendmplib	*table*	\everyendmplib
everymplib	*table*	\everymplib
getcachedir	*function* (<*string*>)	\mplibcachedir
globaltextext	*boolean*	\mplibglobaltextext
legacyverbatimtex	*boolean*	\mpliblegacybehavior
noneedtoreplace	*table*	\mplibmakenocache
numbersystem	*string*	\mplibnumbersystem
setformat	*function* (<*string*>)	\mplibsetformat
showlog	*boolean*	\mplibshowlog
textextlabel	*boolean*	\mplibtextextlabel
verbatiminput	*boolean*	\mplibverbatim

```
  boolean b; b = 1 > 2;
  numeric n; n = 3;
  string s; s = "MetaPost";
  path p; p = unitsquare;
\end{mplibcode}

\directlua{
  local instance1 = luamplib.instances.instance1
  print( instance1:get_boolean "b" )
  print( instance1:get_number  "n" )
  print( instance1:get_string  "s" )
  local t = instance1:get_path "p"
  for k,v in pairs(t) do
    print(k, type(v)=='table' and table.concat(v,' ') or v)
  end
}
```

Lua function `luamplib.process_mplibcode` Users can execute a MetaPost code chunk from Lua side by using this function:

```
luamplib.process_mplibcode (<string> metapost code, <string> instance name)
```

The second argument cannot be absent, but can be an empty string ("") which means that it has no instance name.

Some other elements in the luamplib namespace, listed in Table 2, can have effects on the process of process_mplibcode.

2 Implementation

2.1 Lua module

```
1
2 luatexbase.provides_module {
3   name          = "luamplib",
4   version       = "2.34.0",
```

```
  date          = "2024/07/17",
  description   = "Lua package to typeset Metapost with LuaTeX's MPLib.",
}

```

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

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

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

```

Use our own function for warn/info/err.

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

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

luamplib.showlog  = luamplib.showlog or false

```

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

```
local tableconcat = table.concat
local tableinsert = table.insert
```

```
local tableunpack = table.unpack
local texsprint   = tex.sprint
local texgettoks  = tex.gettoks
local texgetbox   = tex.getbox
local texruntoks  = tex.runtoks

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

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

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

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

```

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

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

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

```

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

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

```

```
local currenttime = os.time()

local outputdir, cachedir
if lfstouch then
  for i,v in ipairs{'TEXMFVAR','TEXMF_OUTPUT_DIRECTORY','.','TEXMFOUTPUT'} do
    local var = i == 3 and v or kpse.var_value(v)
    if var and var ~= "" then
      for _,vv in next, var:explode(os.type == "unix" and ":" or ";") do
        local dir = format("%s/%s",vv,"luamplib_cache")
        if not lfsisdir(dir) then
          mk_full_path(dir)
        end
        if is_writable(dir) then
          outputdir = dir
          break
        end
      end
      if outputdir then break end
    end
  end
end
outputdir = outputdir or '.'
function luamplib.getcachedir(dir)
  dir = dir:gsub("##","#")
  dir = dir:gsub("^~",
    os.type == "windows" and os.getenv("UserProfile") or os.getenv("HOME"))
  if lfstouch and dir then
    if lfsisdir(dir) then
      if is_writable(dir) then
        cachedir = dir
      else
        warn("Directory '%s' is not writable!", dir)
      end
    else
      warn("Directory '%s' does not exist!", dir)
    end
  end
end

```

Some basic MetaPost files not necessary to make cache files.

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

```
  ["mp-tool.mpiv"] = true, ["mp-cont.mpiv"] = true,
}
luamplib.noneedtoreplace = noneedtoreplace

```

format.mp is much complicated, so specially treated.

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

```

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

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

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

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

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

```

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

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

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

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

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

```

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

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

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

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

```
    end
    return file
  end
end

```

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.)

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

```

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

```
local currentformat = "plain"
function luamplib.setformat (name)
  currentformat = name
end

```

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

```
luamplib.codeinherit = false
local mplibinstances = {}
luamplib.instances = mplibinstances
local has_instancename = false

local function reporterror (result, prevlog)
  if not result then
    err("no result object returned")
  else
    local t, e, l = result.term, result.error, result.log
```

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

```
    local log = l or t or "no-term"
    log = log:gsub("%(Please type a command or say `end'%)",""):gsub("\n+","\n")
    if result.status > 0 then
      local first = log:match"(.-\n! .-)\n! "
      if first then
        termorlog("term", first)
        termorlog("log", log, "Warning")
      else
        warn(log)
      end
      if result.status > 1 then
        err(e or "see above messages")
      end
    elseif prevlog then
      log = prevlog..log
```

v2.6.1: now luamplib does not disregard `show` command, even when `luamplib.showlog` is false. Incidentally, it does not raise error nor prints an info, even if output has no figure.

```
      local show = log:match"\n>>? .+"
      if show then
```

```
        termorlog("term", show, "Info (more info in the log)")
        info(log)
      elseif luamplib.showlog and log:find"%g" then
        info(log)
      end
    end
    return log
  end
end
```

lualibs-os.lua installs a randomseed. When this file is not loaded, we should explicitly seed a unique integer to get random randomseed for each run.

```
if not math.initialseed then math.randomseed(currenttime) end
local function luamplibload (name)
  local mpx = mplib.new {
    ini_version = true,
    find_file   = luamplib.finder,
```

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

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

Append our own MetaPost preamble to the preamble above.

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

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

```
local function process (data, instancename)
  local currfmt
  if instancename and instancename ~= "" then
    currfmt = instancename
    has_instancename = true
  else
```

```
    currfmt = tableconcat{
      currentformat,
      luamplib.numbersystem or "scaled",
      tostring(luamplib.textextlabel),
      tostring(luamplib.legacyverbatimtex),
    }
    has_instancename = false
  end
  local mpx = mplibinstances[currfmt]
  local standalone = not (has_instancename or luamplib.codeinherit)
  if mpx and standalone then
    mpx:finish()
  end
  local log = ""
  if standalone or not mpx then
    mpx, _, log = luamplibload(currentformat)
    mplibinstances[currfmt] = mpx
  end
  local converted, result = false, {}
  if mpx and data then
    result = mpx:execute(data)
    local log = reporterror(result, log)
    if log then
      if result.fig then
        converted = luamplib.convert(result)
      end
    end
  else
    err"Mem file unloadable. Maybe generated with a different version of mplib?"
  end
  return converted, result
end

```

dvipdfmx is supported, though nobody seems to use it.

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

make_text and some run_script uses LuaTEX's tex.runtoks.

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

```

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

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

```

Prepare textext box number containers, locals and globals. 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 true. Boxes in instances with name will also be global, so that their tex boxes can be shared among instances of the same name.

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

For conversion of sp to bp.

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

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

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

```

Make color or xcolor's color expressions usable, with \mpcolor or mplibcolor. These commands should be used with graphical objects. Attempt to support l3color as well.

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

local colfmt = is_defined'color_select:n' and "l3color" or "xcolor"
if colfmt == "l3color" then
  run_tex_code{
    "\\newcatcodetable\\luamplibcctabexplat",
    "\\begingroup",
    "\\catcode`@=11 ",
    "\\catcode`_=11 ",
```

```
    "\\catcode`:=11 ",
    "\\savecatcodetable\\luamplibcctabexplat",
    "\\endgroup",
  }
end
local ccexplat = luatexbase.registernumber"luamplibcctabexplat"

local function process_color (str)
  if str then
    if not str:find("%b{}") then
      str = format("{%s}",str)
    end
    local myfmt = mplibcolorfmt[colfmt]
    if colfmt == "l3color" and is_defined"color" then
      if str:find("%b[]") then
        myfmt = mplibcolorfmt.xcolor
      else
        for _,v in ipairs(str:match"{(.+)}":explode"!") do
          if not v:find("^%s*%d+%s*$") then
            local pp = get_macro(format("l__color_named_%s_prop",v))
            if not pp or pp == "" then
              myfmt = mplibcolorfmt.xcolor
              break
            end
          end
        end
      end
    end
    run_tex_code(myfmt:format(str), ccexplat or catat11)
    local t = texgettoks"mplibtmptoks"
    if not pdfmode and not t:find"^pdf" then
      t = t:gsub("%a+ (.+)","pdf:bc [%1]")
    end
    return format('1 withprescript "mpliboverridecolor=%s"', t)
  end
  return ""
end
```

for `\mpdim` or `mplibdimen`

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

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

```
local function process_verbatimtex_text (str)
  if str then
    run_tex_code(str)
```

```
  end
  return ""
end

```

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

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

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

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

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

```

For metafun format. see issue #79.

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

```

metafun 2021-03-09 changes crashes luamplib.

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

```

A function from ConTEXt general.

```
local function mpprint(buffer,...)
  for i=1,select("#",...) do
    local value = select(i,...)
    if value ~= nil then
      local t = type(value)
      if t == "number" then
        buffer[#buffer+1] = format("%.16f",value)
      elseif t == "string" then
        buffer[#buffer+1] = value
      elseif t == "table" then
        buffer[#buffer+1] = "(" .. tableconcat(value,",") .. ")"
      else -- boolean or whatever
        buffer[#buffer+1] = tostring(value)
      end
    end
  end
end

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

```

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

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

```
587
588 luamplib.legacyverbatimtex = true
589
590 function luamplib.maketext (str, what)
591   if str and str ~= "" then
592     str = protecttexcontents(str)
593     if what == 1 then
594       if not str:find("\\documentclass"..name_e) and
595          not str:find("\\begin%s*{document}") and
596          not str:find("\\documentstyle"..name_e) and
597          not str:find("\\usepackage"..name_e) then
598         if luamplib.legacyverbatimtex then
599           if luamplib.in_the_fig then
600             return process_verbatimtex_infig(str)
601           else
602             return process_verbatimtex_prefig(str)
603           end
604         else
605           return process_verbatimtex_text(str)
606         end
607       end
608     else
609       return process_tex_text(str)
610     end
611   end
612   return ""
613 end
614
```

luamplib's metapost color operators

```
615 local function colorsplit (res)
616   local t, tt = { }, res:gsub("[%[%]]",""):explode()
617   local be = tt[1]:find"^%d" and 1 or 2
618   for i=be, #tt do
619     if tt[i]:find"^%a" then break end
620     t[#t+1] = tt[i]
621   end
622   return t
623 end
624
625 luamplib.gettexcolor = function (str, rgb)
626   local res = process_color(str):match'"mpliboverridecolor=(.+)"'
627   if res:find" cs " or res:find"@pdf.obj" then
628     if not rgb then
629       warn("%s is a spot color. Forced to CMYK", str)
630     end
631     run_tex_code({
632       "\\color_export:nnN{",
633       str,
634       "}{",
635       rgb and "space-sep-rgb" or "space-sep-cmyk",
636       "}\\mplib_@tempa",
637     },ccexplat)
638     return get_macro"mplib_@tempa":explode()
639   end
```

```
  local t = colorsplit(res)
  if #t == 3 or not rgb then return t end
  if #t == 4 then
    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]) }
  end
  return { t[1], t[1], t[1] }
end

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

An example of spot color shading:

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

```
endfig;
\end{mplibcode}
\end{document}
```

another one: user-defined DeviceN colorspace

```
\DocumentMetadata{ }
\documentclass{article}
\usepackage{luamplib}
\mplibsetformat{metafun}
\ExplSyntaxOn
\color_model_new:nnn { pantone1215 }
  { Separation }
  { name = PANTONE~1215~U ,
    alternative-model = cmyk ,
    alternative-values = {0, 0.15, 0.51, 0}
  }
\color_model_new:nnn { pantone+black }
  { DeviceN }
  {
    names = {pantone1215,black}
  }
\color_set:nnn{purepantone}{pantone+black}{1,0}
\color_set:nnn{pureblack} {pantone+black}{0,1}
\ExplSyntaxOff
\begin{document}
\mpfig
fill unitsquare xscaled \mpdim{\textwidth} yscaled 30
    withshademethod "linear"
    withshadecolors ("purepantone","pureblack")
    ;
\endmpfig
\end{document}
```

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

```

luamplib's mplibgraphictext operator

```
local running = -1073741824
local emboldenfonts = { }
local function getemboldenwidth (curr, fakebold)
  local width = emboldenfonts.width
```

```
  if not width then
    local f
    local function getglyph(n)
      while n do
        if n.head then
          getglyph(n.head)
        elseif n.font and n.font > 0 then
          f = n.font; break
        end
        n = node.getnext(n)
      end
    end
    getglyph(curr)
    width = font.getcopy(f or font.current()).size * fakebold / factor * 10
    emboldenfonts.width = width
  end
  return width
end
local function getrulewhatsit (line, wd, ht, dp)
  line, wd, ht, dp = line/1000, wd/factor, ht/factor, dp/factor
  local pl
  local fmt = "%f w %f %f %f %f re %s"
  if pdfmode then
    pl = node.new("whatsit","pdf_literal")
    pl.mode = 0
  else
    fmt = "pdf:content "..fmt
    pl = node.new("whatsit","special")
  end
  pl.data = fmt:format(line, 0, -dp, wd, ht+dp, "B")
  local ss = node.new"glue"
  node.setglue(ss, 0, 65536, 65536, 2, 2)
  pl.next = ss
  return pl
end
local function getrulemetric (box, curr, bp)
  local wd,ht,dp = curr.width, curr.height, curr.depth
  wd = wd == running and box.width  or wd
  ht = ht == running and box.height or ht
  dp = dp == running and box.depth  or dp
  if bp then
    return wd/factor, ht/factor, dp/factor
  end
  return wd, ht, dp
end
local function embolden (box, curr, fakebold)
  local head = curr
  while curr do
    if curr.head then
      curr.head = embolden(curr, curr.head, fakebold)
    elseif curr.replace then
      curr.replace = embolden(box, curr.replace, fakebold)
    elseif curr.leader then
      if curr.leader.head then
```

```
      curr.leader.head = embolden(curr.leader, curr.leader.head, fakebold)
    elseif curr.leader.id == node.id"rule" then
      local glue = node.effective_glue(curr, box)
      local line = getemboldenwidth(curr, fakebold)
      local wd,ht,dp = getrulemetric(box, curr.leader)
      if box.id == node.id"hlist" then
        wd = glue
      else
        ht, dp = 0, glue
      end
      local pl = getrulewhatsit(line, wd, ht, dp)
      local pack = box.id == node.id"hlist" and node.hpack or node.vpack
      local list = pack(pl, glue, "exactly")
      head = node.insert_after(head, curr, list)
      head, curr = node.remove(head, curr)
    end
  elseif curr.id == node.id"rule" and curr.subtype == 0 then
    local line = getemboldenwidth(curr, fakebold)
    local wd,ht,dp = getrulemetric(box, curr)
    if box.id == node.id"vlist" then
      ht, dp = 0, ht+dp
    end
    local pl = getrulewhatsit(line, wd, ht, dp)
    local list
    if box.id == node.id"hlist" then
      list = node.hpack(pl, wd, "exactly")
    else
      list = node.vpack(pl, ht+dp, "exactly")
    end
    head = node.insert_after(head, curr, list)
    head, curr = node.remove(head, curr)
  elseif curr.id == node.id"glyph" and curr.font > 0 then
    local f = curr.font
    local i = emboldenfonts[f]
    if not i then
      local ft = font.getfont(f) or font.getcopy(f)
      if pdfmode then
        width = ft.size * fakebold / factor * 10
        emboldenfonts.width = width
        ft.mode, ft.width = 2, width
        i = font.define(ft)
      else
        if ft.format ~= "opentype" and ft.format ~= "truetype" then
          goto skip_type1
        end
        local name = ft.name:gsub('"','''):gsub(';$','')
        name = format('%s;embolden=%s;',name,fakebold)
        _, i = fonts.constructors.readanddefine(name,ft.size)
      end
      emboldenfonts[f] = i
    end
    curr.font = i
  end
  ::skip_type1::
```

```
778     curr = node.getnext(curr)
779   end
780   return head
781 end
782 local function graphictextcolor (col, filldraw)
783   if col:find"^[%d%.:]+$" then
784     col = col:explode":"
785     if pdfmode then
786       local op = #col == 4 and "k" or #col == 3 and "rg" or "g"
787       col[#col+1] = filldraw == "fill" and op or op:upper()
788       return tableconcat(col," ")
789     end
790     return format("[%s]", tableconcat(col," "))
791   end
792   col = process_color(col):match'"mpliboverridecolor=(.+)"'
793   if pdfmode then
794     local t, tt = col:explode(), { }
795     local b = filldraw == "fill" and 1 or #t/2+1
796     local e = b == 1 and #t/2 or #t
797     for i=b,e do
798       tt[#tt+1] = t[i]
799     end
800     return tableconcat(tt," ")
801   end
802   return col:gsub("^.- ","")
803 end
804 luamplib.graphictext = function (text, fakebold, fc, dc)
805   local fmt = process_tex_text(text):sub(1,-2)
806   local id = tonumber(fmt:match"mplibtexboxid=(%d+):")
807   emboldenfonts.width = nil
808   local box = texgetbox(id)
809   box.head = embolden(box, box.head, fakebold)
810   local fill = graphictextcolor(fc,"fill")
811   local draw = graphictextcolor(dc,"draw")
812   local bc = pdfmode and "" or "pdf:bc "
813   return format('%s withprescript "mpliboverridecolor=%s%s %s")', fmt, bc, fill, draw)
814 end
815
```

luamplib's mplibglyph operator

```
816 local function mperr (str)
817   return format("hide(errmessage %q)", str)
818 end
819 local function getangle (a,b,c)
820   local r = math.deg(math.atan(c.y-b.y, c.x-b.x) - math.atan(b.y-a.y, b.x-a.x))
821   if r > 180 then
822     r = r - 360
823   elseif r < -180 then
824     r = r + 360
825   end
826   return r
827 end
828 local function turning (t)
829   local r, n = 0, #t
830   for i=1,2 do
```

```
    tableinsert(t, t[i])
  end
  for i=1,n do
    r = r + getangle(t[i], t[i+1], t[i+2])
  end
  return r/360
end
local function glyphimage(t, fmt)
  local q,p,r = {{},{}}
  for i,v in ipairs(t) do
    local cmd = v[#v]
    if cmd == "m" then
      p = {format('(%s,%s)',v[1],v[2])}
      r = {{x=v[1],y=v[2]}}
    else
      local nt = t[i+1]
      local last = not nt or nt[#nt] == "m"
      if cmd == "l" then
        local pt = t[i-1]
        local seco = pt[#pt] == "m"
        if (last or seco) and r[1].x == v[1] and r[1].y == v[2] then
        else
          tableinsert(p, format('--(%s,%s)',v[1],v[2]))
          tableinsert(r, {x=v[1],y=v[2]})
        end
        if last then
          tableinsert(p, '--cycle')
        end
      elseif cmd == "c" then
        tableinsert(p, format('..controls(%s,%s)and(%s,%s)',v[1],v[2],v[3],v[4]))
        if last and r[1].x == v[5] and r[1].y == v[6] then
          tableinsert(p, '..cycle')
        else
          tableinsert(p, format('..(%s,%s)',v[5],v[6]))
          if last then
            tableinsert(p, '--cycle')
          end
          tableinsert(r, {x=v[5],y=v[6]})
        end
      else
        return mperr"unknown operator"
      end
      if last then
        tableinsert(q[ turning(r) > 0 and 1 or 2 ], tableconcat(p))
      end
    end
  end
  r = { }
  if fmt == "opentype" then
    for _,v in ipairs(q[1]) do
      tableinsert(r, format('addto currentpicture contour %s;',v))
    end
    for _,v in ipairs(q[2]) do
      tableinsert(r, format('addto currentpicture contour %s withcolor background;',v))
```

```
    end
  else
    for _,v in ipairs(q[2]) do
      tableinsert(r, format('addto currentpicture contour %s;',v))
    end
    for _,v in ipairs(q[1]) do
      tableinsert(r, format('addto currentpicture contour %s withcolor background;',v))
    end
  end
  return format('image(%s)', tableconcat(r))
end
if not table.tofile then require"lualibs-lpeg"; require"lualibs-table"; end
function luamplib.glyph (f, c)
  local filename, subfont, instance, kind, shapedata
  local fid = tonumber(f) or font.id(f)
  if fid > 0 then
    local fontdata = font.getfont(fid) or font.getcopy(fid)
    filename, subfont, kind = fontdata.filename, fontdata.subfont, fontdata.format
    instance = fontdata.specification and fontdata.specification.instance
    filename = filename and filename:gsub("^harfloaded:","")
  else
    local name
    f = f:match"^%s*(.+)%s*$"
    name, subfont, instance = f:match"(.+)%((%d+)%)%[(.-)%]$"
    if not name then
      name, instance = f:match"(.+)%[(.-)%]$" -- SourceHanSansK-VF.otf[Heavy]
    end
    if not name then
      name, subfont = f:match"(.+)%((%d+)%)$" -- Times.ttc(2)
    end
    name = name or f
    subfont = (subfont or 0)+1
    instance = instance and instance:lower()
    for _,ftype in ipairs{"opentype", "truetype"} do
      filename = kpse.find_file(name, ftype.." fonts")
      if filename then
        kind = ftype; break
      end
    end
  end
  if kind ~= "opentype" and kind ~= "truetype" then
    f = fid and fid > 0 and tex.fontname(fid) or f
    if kpse.find_file(f, "tfm") then
      return format("glyph %s of %q", tonumber(c) or format("%q",c), f)
    else
      return mperr"font not found"
    end
  end
  local time = lfsattributes(filename,"modification")
  local k = format("shapes_%s(%s)[%s]", filename, subfont or "", instance or "")
  local h = format(string.rep('%02x', 256/8), string.byte(sha2.digest256(k), 1, -1))
  local newname = format("%s/%s.lua", cachedir or outputdir, h)
  local newtime = lfsattributes(newname,"modification") or 0
  if time == newtime then
```

```
    shapedata = require(newname)
  end
  if not shapedata then
    shapedata = fonts and fonts.handlers.otf.readers.loadshapes(filename,subfont,instance)
    if not shapedata then return mperr"loadshapes() failed. luaotfload not loaded?" end
    table.tofile(newname, shapedata, "return")
    lfstouch(newname, time, time)
  end
  local gid = tonumber(c)
  if not gid then
    local uni = utf8.codepoint(c)
    for i,v in pairs(shapedata.glyphs) do
      if c == v.name or uni == v.unicode then
        gid = i; break
      end
    end
  end
  if not gid then return mperr"cannot get GID (glyph id)" end
  local fac = 1000 / (shapedata.units or 1000)
  local t = shapedata.glyphs[gid].segments
  if not t then return "image()" end
  for i,v in ipairs(t) do
    if type(v) == "table" then
      for ii,vv in ipairs(v) do
        if type(vv) == "number" then
          t[i][ii] = format("%.0f", vv * fac)
        end
      end
    end
  end
  kind = shapedata.format or kind
  return glyphimage(t, kind)
end

```

mpliboutlinetext : based on mkiv's font-mps.lua

```
local rulefmt = "mpliboutlinepic[%i]:=image(addto currentpicture contour \z
unitsquare shifted - center unitsquare;) xscaled %f yscaled %f shifted (%f,%f);"
local outline_horz, outline_vert
function outline_vert (res, box, curr, xshift, yshift)
  local b2u = box.dir == "LTL"
  local dy = (b2u and -box.depth or box.height)/factor
  local ody = dy
  while curr do
    if curr.id == node.id"rule" then
      local wd, ht, dp = getrulemetric(box, curr, true)
      local hd = ht + dp
      if hd ~= 0 then
        dy = dy + (b2u and dp or -ht)
        if wd ~= 0 and curr.subtype == 0 then
          res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+wd/2, yshift+dy+(ht-dp)/2)
        end
        dy = dy + (b2u and ht or -dp)
      end
    elseif curr.id == node.id"glue" then
```

```
    local vwidth = node.effective_glue(curr,box)/factor
    if curr.leader then
      local curr, kind = curr.leader, curr.subtype
      if curr.id == node.id"rule" then
        local wd = getrulemetric(box, curr, true)
        if wd ~= 0 then
          local hd = vwidth
          local dy = dy + (b2u and 0 or -hd)
          if hd ~= 0 and curr.subtype == 0 then
            res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+wd/2, yshift+dy+hd/2)
          end
        end
      elseif curr.head then
        local hd = (curr.height + curr.depth)/factor
        if hd <= vwidth then
          local dy, n, iy = dy, 0, 0
          if kind == 100 or kind == 103 then -- todo: gleaders
            local ady = abs(ody - dy)
            local ndy = math.ceil(ady / hd) * hd
            local diff = ndy - ady
            n = (vwidth-diff) // hd
            dy = dy + (b2u and diff or -diff)
          else
            n = vwidth // hd
            if kind == 101 then
              local side = vwidth % hd / 2
              dy = dy + (b2u and side or -side)
            elseif kind == 102 then
              iy = vwidth % hd / (n+1)
              dy = dy + (b2u and iy or -iy)
            end
          end
          dy = dy + (b2u and curr.depth or -curr.height)/factor
          hd = b2u and hd or -hd
          iy = b2u and iy or -iy
          local func = curr.id == node.id"hlist" and outline_horz or outline_vert
          for i=1,n do
            res = func(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
            dy = dy + hd + iy
          end
        end
      end
    end
    dy = dy + (b2u and vwidth or -vwidth)
  elseif curr.id == node.id"kern" then
    dy = dy + curr.kern/factor * (b2u and 1 or -1)
  elseif curr.id == node.id"vlist" then
    dy = dy + (b2u and curr.depth or -curr.height)/factor
    res = outline_vert(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
    dy = dy + (b2u and curr.height or -curr.depth)/factor
  elseif curr.id == node.id"hlist" then
    dy = dy + (b2u and curr.depth or -curr.height)/factor
    res = outline_horz(res, curr, curr.head, xshift+curr.shift/factor, yshift+dy)
    dy = dy + (b2u and curr.height or -curr.depth)/factor
```

```
    end
    curr = node.getnext(curr)
  end
  return res
end
function outline_horz (res, box, curr, xshift, yshift, discwd)
  local r2l = box.dir == "TRT"
  local dx = r2l and (discwd or box.width/factor) or 0
  local dirs = { { dir = r2l, dx = dx } }
  while curr do
    if curr.id == node.id"dir" then
      local sign, dir = curr.dir:match"(.)(...)"
      local level, newdir = curr.level, r2l
      if sign == "+" then
        newdir = dir == "TRT"
        if r2l ~= newdir then
          local n = node.getnext(curr)
          while n do
            if n.id == node.id"dir" and n.level+1 == level then break end
            n = node.getnext(n)
          end
          n = n or node.tail(curr)
          dx = dx + node.rangedimensions(box, curr, n)/factor * (newdir and 1 or -1)
        end
        dirs[level] = { dir = r2l, dx = dx }
      else
        local level = level + 1
        newdir = dirs[level].dir
        if r2l ~= newdir then
          dx = dirs[level].dx
        end
      end
      r2l = newdir
    elseif curr.char and curr.font and curr.font > 0 then
      local ft = font.getfont(curr.font) or font.getcopy(curr.font)
      local gid = ft.characters[curr.char].index or curr.char
      local scale = ft.size / factor / 1000
      local slant   = (ft.slant or 0)/1000
      local extend  = (ft.extend or 1000)/1000
      local squeeze = (ft.squeeze or 1000)/1000
      local expand  = 1 + (curr.expansion_factor or 0)/1000000
      local xscale = scale * extend * expand
      local yscale = scale * squeeze
      dx = dx - (r2l and curr.width/factor*expand or 0)
      local xpos = dx + xshift + (curr.xoffset or 0)/factor
      local ypos = yshift + (curr.yoffset or 0)/factor
      local vertical = ft.shared and ft.shared.features.vertical and "rotated 90" or ""
      if vertical ~= "" then -- luatexko
        for _,v in ipairs(ft.characters[curr.char].commands or { }) do
          if v[1] == "down" then
            ypos = ypos - v[2] / factor
          elseif v[1] == "right" then
            xpos = xpos + v[2] / factor
          else
```

```
1100            break
1101          end
1102        end
1103      end
1104      local image
1105      if ft.format == "opentype" or ft.format == "truetype" then
1106        image = luamplib.glyph(curr.font, gid)
1107      else
1108        local name, scale = ft.name, 1
1109        local vf = font.read_vf(name, ft.size)
1110        if vf and vf.characters[gid] then
1111          local cmds = vf.characters[gid].commands or {}
1112          for _,v in ipairs(cmds) do
1113            if v[1] == "char" then
1114              gid = v[2]
1115            elseif v[1] == "font" and vf.fonts[v[2]] then
1116              name  = vf.fonts[v[2]].name
1117              scale = vf.fonts[v[2]].size / ft.size
1118            end
1119          end
1120        end
1121        image = format("glyph %s of %q scaled %f", gid, name, scale)
1122      end
1123      res[#res+1] = format("mpliboutlinepic[%i]:=%s xscaled %f yscaled %f slanted %f %s shifted (%f,%f);",
1124                           #res+1, image, xscale, yscale, slant, vertical, xpos, ypos)
1125      dx = dx + (r2l and 0 or curr.width/factor*expand)
1126    elseif curr.replace then
1127      local width = node.dimensions(curr.replace)/factor
1128      dx = dx - (r2l and width or 0)
1129      res = outline_horz(res, box, curr.replace, xshift+dx, yshift, width)
1130      dx = dx + (r2l and 0 or width)
1131    elseif curr.id == node.id"rule" then
1132      local wd, ht, dp = getrulemetric(box, curr, true)
1133      if wd ~= 0 then
1134        local hd = ht + dp
1135        dx = dx - (r2l and wd or 0)
1136        if hd ~= 0 and curr.subtype == 0 then
1137          res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
1138        end
1139        dx = dx + (r2l and 0 or wd)
1140      end
1141    elseif curr.id == node.id"glue" then
1142      local width = node.effective_glue(curr, box)/factor
1143      dx = dx - (r2l and width or 0)
1144      if curr.leader then
1145        local curr, kind = curr.leader, curr.subtype
1146        if curr.id == node.id"rule" then
1147          local wd, ht, dp = getrulemetric(box, curr, true)
1148          local hd = ht + dp
1149          if hd ~= 0 then
1150            wd = width
1151            if wd ~= 0 and curr.subtype == 0 then
1152              res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
1153            end
```

```
1154          end
1155        elseif curr.head then
1156          local wd = curr.width/factor
1157          if wd <= width then
1158            local dx = r2l and dx+width or dx
1159            local n, ix = 0, 0
1160            if kind == 100 or kind == 103 then -- todo: gleaders
1161              local adx = abs(dx-dirs[1].dx)
1162              local ndx = math.ceil(adx / wd) * wd
1163              local diff = ndx - adx
1164              n = (width-diff) // wd
1165              dx = dx + (r2l and -diff-wd or diff)
1166            else
1167              n = width // wd
1168              if kind == 101 then
1169                local side = width % wd /2
1170                dx = dx + (r2l and -side-wd or side)
1171              elseif kind == 102 then
1172                ix = width % wd / (n+1)
1173                dx = dx + (r2l and -ix-wd or ix)
1174              end
1175            end
1176            wd = r2l and -wd or wd
1177            ix = r2l and -ix or ix
1178            local func = curr.id == node.id"hlist" and outline_horz or outline_vert
1179            for i=1,n do
1180              res = func(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1181              dx = dx + wd + ix
1182            end
1183          end
1184        end
1185      end
1186      dx = dx + (r2l and 0 or width)
1187    elseif curr.id == node.id"kern" then
1188      dx = dx + curr.kern/factor * (r2l and -1 or 1)
1189    elseif curr.id == node.id"math" then
1190      dx = dx + curr.surround/factor * (r2l and -1 or 1)
1191    elseif curr.id == node.id"vlist" then
1192      dx = dx - (r2l and curr.width/factor or 0)
1193      res = outline_vert(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1194      dx = dx + (r2l and 0 or curr.width/factor)
1195    elseif curr.id == node.id"hlist" then
1196      dx = dx - (r2l and curr.width/factor or 0)
1197      res = outline_horz(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
1198      dx = dx + (r2l and 0 or curr.width/factor)
1199    end
1200    curr = node.getnext(curr)
1201  end
1202  return res
1203 end
1204 function luamplib.outlinetext (text)
1205   local fmt = process_tex_text(text)
1206   local id  = tonumber(fmt:match"mplibtexboxid=(%d+):")
1207   local box = texgetbox(id)
```

```
  local res = outline_horz({ }, box, box.head, 0, 0)
  if #res == 0 then res = { "mpliboutlinepic[1]:=image();" } end
  return tableconcat(res) .. format("mpliboutlinenum:=%i;", #res)
end

```

Our MetaPost preambles

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

```
def mplibrgbtexcolor primary c =
  runscript("return luamplib.gettexcolor('"& c &"','rgb')")
enddef;
def mplibgraphictext primary t =
  begingroup;
  mplibgraphictext_ (t)
enddef;
def mplibgraphictext_ (expr t) text rest =
  save fakebold, scale, fillcolor, drawcolor, withfillcolor, withdrawcolor,
    fb, fc, dc, graphictextpic;
  picture graphictextpic; graphictextpic := nullpicture;
  numeric fb; string fc, dc; fb:=2; fc:="white"; dc:="black";
  let scale = scaled;
  def fakebold  primary c = hide(fb:=c;) enddef;
  def fillcolor primary c = hide(fc:=colordecimals c;) enddef;
  def drawcolor primary c = hide(dc:=colordecimals c;) enddef;
  let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
  addto graphictextpic doublepath origin rest; graphictextpic:=nullpicture;
  def fakebold  primary c = enddef;
  let fillcolor = fakebold; let drawcolor = fakebold;
  let withfillcolor = fillcolor; let withdrawcolor = drawcolor;
  image(draw runscript("return luamplib.graphictext([===["&t&"]===],"
    & decimal fb &",'"& fc &"','"& dc &"')") rest;)
  endgroup;
enddef;
def mplibglyph expr c of f =
  runscript (
    "return luamplib.glyph('"
    & if numeric f: decimal fi f
    & "','"
    & if numeric c: decimal fi c
    & "')"
  )
enddef;
def mplibdrawglyph expr g =
  draw image(
    save i; numeric i; i:=0;
    for item within g:
      i := i+1;
      fill pathpart item
      if i < length g: withpostscript "collect" fi;
    endfor
  )
enddef;
def mplib_do_outline_text_set_b (text f) (text d) text r =
  def mplib_do_outline_options_f = f enddef;
  def mplib_do_outline_options_d = d enddef;
  def mplib_do_outline_options_r = r enddef;
enddef;
def mplib_do_outline_text_set_f (text f) text r =
  def mplib_do_outline_options_f = f enddef;
  def mplib_do_outline_options_r = r enddef;
enddef;
def mplib_do_outline_text_set_u (text f) text r =
```

```
  def mplib_do_outline_options_f = f enddef;
enddef;
def mplib_do_outline_text_set_d (text d) text r =
  def mplib_do_outline_options_d = d enddef;
  def mplib_do_outline_options_r = r enddef;
enddef;
def mplib_do_outline_text_set_r (text d) (text f) text r =
  def mplib_do_outline_options_d = d enddef;
  def mplib_do_outline_options_f = f enddef;
  def mplib_do_outline_options_r = r enddef;
enddef;
def mplib_do_outline_text_set_n text r =
  def mplib_do_outline_options_r = r enddef;
enddef;
def mplib_do_outline_text_set_p = enddef;
def mplib_fill_outline_text =
  for n=1 upto mpliboutlinenum:
    i:=0;
    for item within mpliboutlinepic[n]:
      i:=i+1;
      fill pathpart item mplib_do_outline_options_f withpen pencircle scaled 0
      if (n<mpliboutlinenum) or (i<length mpliboutlinepic[n]): withpostscript "collect"; fi
    endfor
  endfor
enddef;
def mplib_draw_outline_text =
  for n=1 upto mpliboutlinenum:
    for item within mpliboutlinepic[n]:
      draw pathpart item mplib_do_outline_options_d;
    endfor
  endfor
enddef;
def mplib_filldraw_outline_text =
  for n=1 upto mpliboutlinenum:
    i:=0;
    for item within mpliboutlinepic[n]:
      i:=i+1;
      if (n<mpliboutlinenum) or (i<length mpliboutlinepic[n]):
        fill pathpart item mplib_do_outline_options_f withpostscript "collect";
      else:
        draw pathpart item mplib_do_outline_options_f withpostscript "both";
      fi
    endfor
  endfor
enddef;
vardef mpliboutlinetext@# (expr t) text rest =
  save kind; string kind; kind := str @#;
  save i; numeric i;
  picture mpliboutlinepic[]; numeric mpliboutlinenum;
  def mplib_do_outline_options_d = enddef;
  def mplib_do_outline_options_f = enddef;
  def mplib_do_outline_options_r = enddef;
  runscript("return luamplib.outlinetext[===["&t&"]===]");
  image ( addto currentpicture also image (
```

```
    if kind = "f":
      mplib_do_outline_text_set_f rest;
      mplib_fill_outline_text;
    elseif kind = "d":
      mplib_do_outline_text_set_d rest;
      mplib_draw_outline_text;
    elseif kind = "b":
      mplib_do_outline_text_set_b rest;
      mplib_fill_outline_text;
      mplib_draw_outline_text;
    elseif kind = "u":
      mplib_do_outline_text_set_u rest;
      mplib_filldraw_outline_text;
    elseif kind = "r":
      mplib_do_outline_text_set_r rest;
      mplib_draw_outline_text;
      mplib_fill_outline_text;
    elseif kind = "p":
      mplib_do_outline_text_set_p;
      mplib_draw_outline_text;
    else:
      mplib_do_outline_text_set_n rest;
      mplib_fill_outline_text;
    fi;
  ) mplib_do_outline_options_r; )
enddef ;
primarydef t withpattern p =
  image( fill t withprescript "mplibpattern=" & if numeric p: decimal fi p; )
enddef;
vardef mplibtransformmatrix (text e) =
  save t; transform t;
  t = identity e;
  runscript("luamplib.transformmatrix = {"
  & decimal xxpart t & ","
  & decimal yxpart t & ","
  & decimal xypart t & ","
  & decimal yypart t & ","
  & decimal xpart  t & ","
  & decimal ypart  t & ","
  & "}");
enddef;
primarydef p withfademethod s =
  if picture p:
    image(
      draw p;
      draw center p withprescript "mplibfadestate=stop";
    )
  else:
    p withprescript "mplibfadestate=stop"
  fi
    withprescript "mplibfadetype=" & s
    withprescript "mplibfadebbox=" &
      decimal xpart llcorner p & ":" &
      decimal ypart llcorner p & ":" &
```

```
      decimal xpart urcorner p & ":" &
      decimal ypart urcorner p
enddef;
def withfadeopacity (expr a,b) =
  withprescript "mplibfadeopacity=" &
    decimal a & ":" &
    decimal b
enddef;
def withfadevector (expr a,b) =
  withprescript "mplibfadevector=" &
    decimal xpart a & ":" &
    decimal ypart a & ":" &
    decimal xpart b & ":" &
    decimal ypart b
enddef;
let withfadecenter = withfadevector;
def withfaderadius (expr a,b) =
  withprescript "mplibfaderadius=" &
    decimal a & ":" &
    decimal b
enddef;
def withfadebbox (expr a,b) =
  withprescript "mplibfadebbox=" &
    decimal xpart a & ":" &
    decimal ypart a & ":" &
    decimal xpart b & ":" &
    decimal ypart b
enddef;
]],
  legacyverbatimtex = [[
def specialVerbatimTeX (text t) = runscript("luamplibprefig{"&t&"}") enddef;
def normalVerbatimTeX  (text t) = runscript("luamplibinfig{"&t&"}") enddef;
let VerbatimTeX = specialVerbatimTeX;
extra_beginfig := extra_beginfig & " let VerbatimTeX = normalVerbatimTeX;"&
  "runscript(" &ditto& "luamplib.in_the_fig=true" &ditto& ");";
extra_endfig := extra_endfig & " let VerbatimTeX = specialVerbatimTeX;"&
  "runscript(" &ditto&
  "if luamplib.in_the_fig then luamplib.figid=luamplib.figid+1 end "&
  "luamplib.in_the_fig=false" &ditto& ");";
]],
  textextlabel = [[
primarydef s infont f = rawtextext(s) enddef;
def fontsize expr f =
  begingroup
  save size; numeric size;
  size := mplibdimen("1em");
  if size = 0: 10pt else: size fi
  endgroup
enddef;
]],
}
```

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

```
luamplib.verbatiminput = false
```

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

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

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

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

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

This is needed for legacy behavior

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

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

These five lines are needed for `mplibverbatim` mode.

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

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

```
  else
    data = data:gsub(btex_etex, function(str)
      return format("btex %s etex ", protect_expansion(str)) -- space
    end)
```

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

Next line to address issue #55

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

  process(data, instancename)
end

```

For parsing prescript materials.

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

```

pdfliterals will be stored in figcontents table, and written to pdf in one go at the end of the flushing figure. Subtable post is for the legacy behavior.

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

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

```
  put2output("\\mplibstarttoPDF{%f}{%f}{%f}{%f}",llx,lly,urx,ury)
end

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

```

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

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

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

```

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

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

```
end

```

Colors

```
local prev_override_color
local function do_preobj_CR(object,prescript)
  if object.postscript == "collect" then return end
  local override = prescript and prescript.mpliboverridecolor
  if override then
    if pdfmode then
      pdf_literalcode(override)
      override = nil
    else
      put2output("\\special{%s}",override)
      prev_override_color = override
    end
  else
    local cs = object.color
    if cs and #cs > 0 then
      pdf_literalcode(luamplib.colorconverter(cs))
      prev_override_color = nil
    elseif not pdfmode then
      override = prev_override_color
      if override then
        put2output("\\special{%s}",override)
      end
    end
  end
  return override
end

```

For transparency and shading

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

local function update_pdfobjs (os, stream)
  local key = os
  if stream then key = key..stream end
  local on = pdfobjs[key]
  if on then
    return on,false
  end
  if pdfmode then
    if stream then
      on = pdf.immediateobj("stream",stream,os)
    else
      on = pdf.immediateobj(os)
    end
  else
    on = pdfetcs.cnt or 1
    if stream then
```

```
      texsprint(format("\\special{pdf:stream @mplibpdfobj%s (%s) <<%s>>}",on,stream,os))
    else
      texsprint(format("\\special{pdf:obj @mplibpdfobj%s %s}",on,os))
    end
    pdfetcs.cnt = on + 1
  end
  pdfobjs[key] = on
  return on,true
end
pdfetcs.resfmt = pdfmode and "%s 0 R" or "@mplibpdfobj%s"

if pdfmode then
  pdfetcs.getpageres = pdf.getpageresources or function() return pdf.pageresources end
  local getpageres = pdfetcs.getpageres
  local setpageres = pdf.setpageresources or function(s) pdf.pageresources = s end
  local initialize_resources = function (name)
    local tabname = format("%s_res",name)
    pdfetcs[tabname] = { }
    if luatexbase.callbacktypes.finish_pdffile then -- ltluatex
      local obj = pdf.reserveobj()
      setpageres(format("%s/%s %i 0 R", getpageres() or "", name, obj))
      luatexbase.add_to_callback("finish_pdffile", function()
        pdf.immediateobj(obj, format("<<%s>>", tableconcat(pdfetcs[tabname])))
      end,
      format("luamplib.%s.finish_pdffile",name))
    end
  end
  pdfetcs.fallback_update_resources = function (name, res)
    local tabname = format("%s_res",name)
    if not pdfetcs[tabname] then
      initialize_resources(name)
    end
    if luatexbase.callbacktypes.finish_pdffile then
      local t = pdfetcs[tabname]
      t[#t+1] = res
    else
      local tpr, n = getpageres() or "", 0
      tpr, n = tpr:gsub(format("/%s<<",name), "%1"..res)
      if n == 0 then
        tpr = format("%s/%s<<%s>>", tpr, name, res)
      end
      setpageres(tpr)
    end
  end
else
  texsprint {
    "\\special{pdf:obj @MPlibTr<<>>}",
    "\\special{pdf:obj @MPlibSh<<>>}",
    "\\special{pdf:obj @MPlibCS<<>>}",
    "\\special{pdf:obj @MPlibPt<<>>}",
  }
  pdfetcs.resadded = { }
end

```

Transparency

```
local transparancy_modes = { [0] = "Normal",
  "Normal",       "Multiply",     "Screen",       "Overlay",
  "SoftLight",    "HardLight",    "ColorDodge",   "ColorBurn",
  "Darken",       "Lighten",      "Difference",   "Exclusion",
  "Hue",          "Saturation",   "Color",        "Luminosity",
  "Compatible",
}
local function add_extgs_resources (on, new)
  local key = format("MPlibTr%s", on)
  if new then
    local val = format(pdfetcs.resfmt, on)
    if pdfmanagement then
      texsprint {
        "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/ExtGState}{", key, "}{", val, "}"
      }
    else
      local tr = format("/%s %s", key, val)
      if is_defined(pdfetcs.pgfextgs) then
        texsprint { "\\csname ", pdfetcs.pgfextgs, "\\endcsname{", tr, "}" }
      elseif pdfmode then
        if is_defined"TRP@list" then
          texsprint(catat11,{
            [[\if@filesw\immediate\write\@auxout{]],
            [[\string\g@addto@macro\string\TRP@list{]],
            tr,
            [[}}\fi]],
          })
          if not get_macro"TRP@list":find(tr) then
            texsprint(catat11,[[\global\TRP@reruntrue]])
          end
        else
          pdfetcs.fallback_update_resources("ExtGState", tr)
        end
      else
        texsprint { "\\special{pdf:put @MPlibTr<<", tr, ">>}" }
      end
    end
  end
  if not pdfmode and not pdfmanagement and not is_defined(pdfetcs.pgfextgs) then
    texsprint"\\special{pdf:put @resources <</ExtGState @MPlibTr>>}"
    pdfetcs.resadded.ExtGState = "@MPlibTr"
  end
  return key
end
local function do_preobj_TR(object,prescript)
  if object.postscript == "collect" then return end
  local opaq = prescript and prescript.tr_transparency
  if opaq then
    local key, on, os, new
    local mode = prescript.tr_alternative or 1
    mode = transparancy_modes[tonumber(mode)] or mode
    for i,v in ipairs{ {mode,opaq},{"Normal",1} } do
      mode, opaq = v[1], v[2]
```

```
      os = format("<</BM /%s/ca %.3f/CA %.3f/AIS false>>",mode,opaq,opaq)
      on, new = update_pdfobjs(os)
      key = add_extgs_resources(on,new)
      if i == 1 then
        pdf_literalcode("/%s gs",key)
      else
        return format("/%s gs",key)
      end
    end
  end
end
```

Shading with metafun format.

```
local function sh_pdfpageresources(shtype,domain,colorspace,ca,cb,coordinates,steps,fractions)
  local fun2fmt,os = "<</FunctionType 2/Domain [%s]/C0 [%s]/C1 [%s]/N 1>>"
  if steps > 1 then
    local list,bounds,encode = { },{ },{ }
    for i=1,steps do
      if i < steps then
        bounds[i] = fractions[i] or 1
      end
      encode[2*i-1] = 0
      encode[2*i]   = 1
      os = fun2fmt:format(domain,tableconcat(ca[i],' '),tableconcat(cb[i],' '))
      list[i] = format(pdfetcs.resfmt, update_pdfobjs(os))
    end
    os = tableconcat {
      "<</FunctionType 3",
      format("/Bounds [%s]",    tableconcat(bounds,' ')),
      format("/Encode [%s]",    tableconcat(encode,' ')),
      format("/Functions [%s]", tableconcat(list,  ' ')),
      format("/Domain [%s]>>",  domain),
    }
  else
    os = fun2fmt:format(domain,tableconcat(ca[1],' '),tableconcat(cb[1],' '))
  end
  local objref = format(pdfetcs.resfmt, update_pdfobjs(os))
  os = tableconcat {
    format("<</ShadingType %i", shtype),
    format("/ColorSpace %s",    colorspace),
    format("/Function %s",      objref),
    format("/Coords [%s]",      coordinates),
    "/Extend [true true]/AntiAlias true>>",
  }
  local on, new = update_pdfobjs(os)
  if new then
    local key, val = format("MPlibSh%s", on), format(pdfetcs.resfmt, on)
    if pdfmanagement then
      texsprint {
        "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/Shading}{", key, "}{", val, "}"
      }
    else
      local res = format("/%s %s", key, val)
      if pdfmode then
```

```
        pdfetcs.fallback_update_resources("Shading", res)
      else
        texsprint { "\\special{pdf:put @MPlibSh<<", res, ">>}" }
      end
    end
  end
  if not pdfmode and not pdfmanagement then
    texsprint"\\special{pdf:put @resources <</Shading @MPlibSh>>}"
    pdfetcs.resadded.Shading = "@MPlibSh"
  end
  return on
end

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

pdfetcs.clrspcs = setmetatable({ }, { __index = function(t,names)
  run_tex_code({
    [[\color_model_new:nnn]],
    format("{mplibcolorspace_%s}", names:gsub(",","_")),
    format("{DeviceN}{names={%s}}", names),
    [[\edef\mplib_@tempa{\pdf_object_ref_last:}]],
  }, ccexplat)
  local colorspace = get_macro'mplib_@tempa'
  t[names] = colorspace
  return colorspace
end })

local function do_preobj_SH(object,prescript)
  local shade_no
  local sh_type = prescript and prescript.sh_type
  if not sh_type then
    return
  else
    local domain  = prescript.sh_domain or "0 1"
    local centera = prescript.sh_center_a or "0 0"; centera = centera:explode()
    local centerb = prescript.sh_center_b or "0 0"; centerb = centerb:explode()
    local transform = prescript.sh_transform == "yes"
    local sx,sy,sr,dx,dy = 1,1,1,0,0
    if transform then
      local first = prescript.sh_first or "0 0"; first = first:explode()
      local setx = prescript.sh_set_x or "0 0";  setx = setx:explode()
      local sety = prescript.sh_set_y or "0 0";  sety = sety:explode()
      local x,y = tonumber(setx[1]) or 0, tonumber(sety[1]) or 0
      if x ~= 0 and y ~= 0 then
```

```
      local path = object.path
      local path1x = path[1].x_coord
      local path1y = path[1].y_coord
      local path2x = path[x].x_coord
      local path2y = path[y].y_coord
      local dxa = path2x - path1x
      local dya = path2y - path1y
      local dxb = setx[2] - first[1]
      local dyb = sety[2] - first[2]
      if dxa ~= 0 and dya ~= 0 and dxb ~= 0 and dyb ~= 0 then
        sx = dxa / dxb ; if sx < 0 then sx = - sx end
        sy = dya / dyb ; if sy < 0 then sy = - sy end
        sr = math.sqrt(sx^2 + sy^2)
        dx = path1x - sx*first[1]
        dy = path1y - sy*first[2]
      end
    end
  end
  local ca, cb, colorspace, steps, fractions
  ca = { prescript.sh_color_a_1 or prescript.sh_color_a or {0} }
  cb = { prescript.sh_color_b_1 or prescript.sh_color_b or {1} }
  steps = tonumber(prescript.sh_step) or 1
  if steps > 1 then
    fractions = { prescript.sh_fraction_1 or 0 }
    for i=2,steps do
      fractions[i] = prescript[format("sh_fraction_%i",i)] or (i/steps)
      ca[i] = prescript[format("sh_color_a_%i",i)] or {0}
      cb[i] = prescript[format("sh_color_b_%i",i)] or {1}
    end
  end
  if prescript.mplib_spotcolor then
    ca, cb = { }, { }
    local names, pos, objref = { }, -1, ""
    local script = object.prescript:explode"\13+"
    for i=#script,1,-1 do
      if script[i]:find"mplib_spotcolor" then
        local t, name, value = script[i]:explode"="[2]:explode":"
        value, objref, name = t[1], t[2], t[3]
        if not names[name] then
          pos = pos+1
          names[name] = pos
          names[#names+1] = name
        end
        t = { }
        for j=1,names[name] do t[#t+1] = 0 end
        t[#t+1] = value
        tableinsert(#ca == #cb and ca or cb, t)
      end
    end
    for _,t in ipairs{ca,cb} do
      for _,tt in ipairs(t) do
        for i=1,#names-#tt do tt[#tt+1] = 0 end
      end
    end
```

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

```

Patterns

```
pdfetcs.patterns = { }
local patterns = pdfetcs.patterns
local function gather_resources (do_pattern, optres)
 local t = { }
 local names = {"ExtGState","ColorSpace","Shading"}
 if do_pattern then
   names[#names+1] = "Pattern"
 end
```

```
1992  if pdfmode then
1993    if pdfmanagement then
1994      for _,v in ipairs(names) do
1995        local pp = get_macro(format("g__pdfdict_/g__pdf_Core/Page/Resources/%s_prop",v))
1996        if pp and pp:find"__prop_pair" then
1997          t[#t+1] = format("/%s %s 0 R", v, ltx.pdf.object_id("__pdf/Page/Resources/"..v))
1998        end
1999      end
2000    else
2001      local res = pdfetcs.getpageres() or ""
2002      run_tex_code[[\mplibtmptoks\expandafter{\the\pdfvariable pageresources}]]
2003      res = (res .. texgettoks'mplibtmptoks'):explode()
2004      res = tableconcat(res," "):explode"/+"
2005      for _,v in ipairs(res) do
2006        if do_pattern or not v:find"Pattern" and not optres:find(v) then
2007          t[#t+1] = "/" .. v
2008        end
2009      end
2010    end
2011  else
2012    if pdfmanagement then
2013      for _,v in ipairs(names) do
2014        local pp = get_macro(format("g__pdfdict_/g__pdf_Core/Page/Resources/%s_prop",v))
2015        if pp and pp:find"__prop_pair" then
2016          run_tex_code {
2017            "\\mplibtmptoks\\expanded{{",
2018            format("/%s \\csname pdf_object_ref:n\\endcsname{__pdf/Page/Resources/%s}",v,v),
2019            "}}",
2020          }
2021          t[#t+1] = texgettoks'mplibtmptoks'
2022        end
2023      end
2024    elseif is_defined(pdfetcs.pgfextgs) then
2025      run_tex_code ({
2026        "\\mplibtmptoks\\expanded{{",
2027        "\\ifpgf@sys@pdf@extgs@exists /ExtGState @pgfextgs\\fi",
2028        "\\ifpgf@sys@pdf@colorspaces@exists /ColorSpace @pgfcolorspaces\\fi",
2029        do_pattern and "\\ifpgf@sys@pdf@patterns@exists /Pattern @pgfpatterns \\fi" or "",
2030        "}}",
2031      }, catat11)
2032      t[#t+1] = texgettoks'mplibtmptoks'
2033    elseif do_pattern then
2034      for _,v in ipairs(names) do
2035        local vv = pdfetcs.resadded[v]
2036        if vv then
2037          t[#t+1] = format("/%s %s", v, vv)
2038        end
2039      end
2040    end
2041  end
2042  return t
2043 end
2044 function luamplib.registerpattern ( boxid, name, opts )
2045   local box = texgetbox(boxid)
```

```
  local wd = format("%.3f",box.width/factor)
  local hd = format("%.3f",(box.height+box.depth)/factor)
  info("w/h/d of '%s': %s %s 0.0", name, wd, hd)
  if opts.xstep == 0 then opts.xstep = nil end
  if opts.ystep == 0 then opts.ystep = nil end
  if opts.colored == nil then
    opts.colored = opts.coloured
    if opts.colored == nil then
      opts.colored = true
    end
  end
  if type(opts.matrix) == "table" then opts.matrix = tableconcat(opts.matrix," ") end
  if type(opts.bbox) == "table" then opts.bbox = tableconcat(opts.bbox," ") end
  if opts.matrix and opts.matrix:find"%a" then
    local data = format("mplibtransformmatrix(%s);",opts.matrix)
    process(data,"@mplibtransformmatrix")
    local t = luamplib.transformmatrix
    opts.matrix = format("%s %s %s %s", t[1], t[2], t[3], t[4])
    opts.xshift = opts.xshift or t[5]
    opts.yshift = opts.yshift or t[6]
  end
  local attr = {
    "/Type/Pattern",
    "/PatternType 1",
    format("/PaintType %i", opts.colored and 1 or 2),
    "/TilingType 2",
    format("/XStep %s", opts.xstep or wd),
    format("/YStep %s", opts.ystep or hd),
    format("/Matrix [%s %s %s]", opts.matrix or "1 0 0 1", opts.xshift or 0, opts.yshift or 0),
  }
  local optres = opts.resources or ""
  local t = gather_resources(false, optres)
  optres = optres .. tableconcat(t)
  if pdfmode then
    if opts.bbox then
      attr[#attr+1] = format("/BBox [%s]", opts.bbox)
    end
    local index = tex.saveboxresource(boxid, tableconcat(attr), optres, true, opts.bbox and 4 or 1)
    patterns[name] = { id = index, colored = opts.colored }
  else
    local objname = "@mplibpattern"..name
    local metric = format("bbox %s", opts.bbox or format("0 0 %s %s",wd,hd))
    texsprint {
      [[\ifvmode\nointerlineskip\fi]],
      format([[\hbox to0pt{\vbox to0pt{\hsize=\wd %i\vss\noindent]], boxid), -- force horiz mode?
      [[\special{pdf:bcontent}]],
      [[\special{pdf:bxobj ]], objname, format(" %s}", metric),
      format([[\raise\dp %i\box %i]], boxid, boxid),
      format([[\special{pdf:put @resources <<%s>>}]], optres),
      [[\special{pdf:exobj <<]], tableconcat(attr), ">>}",
      [[\special{pdf:econtent}]],
      [[\par}\hss}]],
    }
    patterns[#patterns+1] = objname
```

```
2100    patterns[name] = { id = #patterns, colored = opts.colored }
2101  end
2102 end
2103 local function pattern_colorspace (cs)
2104  local on, new = update_pdfobjs(format("[/Pattern %s]", cs))
2105  if new then
2106    local key, val = format("MPlibCS%i",on), format(pdfetcs.resfmt,on)
2107    if pdfmanagement then
2108      texsprint {
2109        "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/ColorSpace}{", key, "}{", val, "}"
2110      }
2111    else
2112      local res = format("/%s %s", key, val)
2113      if is_defined(pdfetcs.pgfcolorspace) then
2114        texsprint { "\\csname ", pdfetcs.pgfcolorspace, "\\endcsname{", res, "}" }
2115      elseif pdfmode then
2116        pdfetcs.fallback_update_resources("ColorSpace", res)
2117      else
2118        texsprint { "\\special{pdf:put @MPlibCS<<", res, ">>}" }
2119      end
2120    end
2121  end
2122  if not pdfmode and not pdfmanagement and not is_defined(pdfetcs.pgfcolorspace) then
2123    texsprint"\\special{pdf:put @resources <</ColorSpace @MPlibCS>>}"
2124    pdfetcs.resadded.ColorSpace = "@MPlibCS"
2125  end
2126  return on
2127 end
2128 local function do_preobj_PAT(object, prescript)
2129  local name = prescript and prescript.mplibpattern
2130  if not name then return end
2131  local patt = patterns[name]
2132  local index = patt and patt.id or err("cannot get pattern object '%s'", name)
2133  local key = format("MPlibPt%s",index)
2134  if patt.colored then
2135    pdf_literalcode("/Pattern cs /%s scn", key)
2136  else
2137    local color = prescript.mpliboverridecolor
2138    if not color then
2139      local t = object.color
2140      color = t and #t>0 and luamplib.colorconverter(t)
2141    end
2142    if not color then return end
2143    local cs
2144    if color:find" cs " or color:find"@pdf.obj" then
2145      local t = color:explode()
2146      if pdfmode then
2147        cs = format("%s 0 R", ltx.pdf.object_id( t[1]:sub(2,-1) ))
2148        color = t[3]
2149      else
2150        cs = t[2]
2151        color = t[3]:match"%[(.+)%]"
2152      end
2153    else
```

```
    local t = colorsplit(color)
    cs = #t == 4 and "/DeviceCMYK" or #t == 3 and "/DeviceRGB" or "/DeviceGray"
    color = tableconcat(t," ")
  end
  pdf_literalcode("/MPlibCS%i cs %s /%s scn", pattern_colorspace(cs), color, key)
end
if not patt.done then
  local val = pdfmode and format("%s 0 R",index) or patterns[index]
  if pdfmanagement then
    texsprint {
      "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/Pattern}{", key, "}{", val, "}"
    }
  else
    local res = format("/%s %s", key, val)
    if is_defined(pdfetcs.pgfpattern) then
      texsprint { "\\csname ", pdfetcs.pgfpattern, "\\endcsname{", res, "}" }
    elseif pdfmode then
      pdfetcs.fallback_update_resources("Pattern", res)
    else
      texsprint { "\\special{pdf:put @MPlibPt<<", res, ">>}" }
    end
  end
end
if not pdfmode and not pdfmanagement and not is_defined(pdfetcs.pgfpattern) then
  texsprint"\\special{pdf:put @resources <</Pattern @MPlibPt>>}"
  pdfetcs.resadded.Pattern = "@MPlibPt"
end
patt.done = true
end

```

Fading

```
pdfetcs.fading = { }
local function do_preobj_FADE (object, prescript)
  local fd_type = prescript and prescript.mplibfadetype
  local fd_stop = prescript and prescript.mplibfadestate
  if not fd_type then
    return fd_stop -- returns "stop" (if picture) or nil
  end
  local bbox = prescript.mplibfadebbox:explode":"
  local dx, dy = -bbox[1], -bbox[2]
  local vec = prescript.mplibfadevector; vec = vec and vec:explode":"
  if not vec then
    if fd_type == "linear" then
      vec = {bbox[1], bbox[2], bbox[3], bbox[2]} -- left to right
    else
      local centerx, centery = (bbox[1]+bbox[3])/2, (bbox[2]+bbox[4])/2
      vec = {centerx, centery, centerx, centery} -- center for both circles
    end
  end
  local coords = { vec[1]+dx, vec[2]+dy, vec[3]+dx, vec[4]+dy }
  if fd_type == "linear" then
    coords = format("%f %f %f %f", tableunpack(coords))
  elseif fd_type == "circular" then
    local width, height = bbox[3]-bbox[1], bbox[4]-bbox[2]
```

```
    local radius = (prescript.mplibfaderadius or "0:"..math.sqrt(width^2+height^2)/2):explode":"
    tableinsert(coords, 3, radius[1])
    tableinsert(coords, radius[2])
    coords = format("%f %f %f %f %f %f", tableunpack(coords))
  else
    err("unknown fading method '%s'", fd_type)
  end
  bbox = format("0 0 %f %f", bbox[3]+dx, bbox[4]+dy)
  fd_type = fd_type == "linear" and 2 or 3
  local opaq = (prescript.mplibfadeopacity or "1:0"):explode":"
  local on, os, new
  on = sh_pdfpageresources(fd_type, "0 1", "/DeviceGray", {{opaq[1]}}, {{opaq[2]}}, coords, 1)
  os = format("<</PatternType 2/Shading %s>>", format(pdfetcs.resfmt, on))
  on = update_pdfobjs(os)
  local streamtext = format("q /Pattern cs/MPlibFd%s scn %s re f Q", on, bbox)
  os = format("<</Pattern<</MPlibFd%s %s>>>>", on, format(pdfetcs.resfmt, on))
  on = update_pdfobjs(os)
  local resources = "/Resources " .. format(pdfetcs.resfmt, on)
  on = update_pdfobjs"<</S/Transparency/CS/DeviceGray>>"
  local attr = tableconcat{
    "/Subtype/Form",
    format("/BBox[%s]", bbox),
    format("/Matrix[1 0 0 1 %f %f]", -dx, -dy),
    resources,
    "/Group ", format(pdfetcs.resfmt, on),
  }
  on = update_pdfobjs(attr, streamtext)
  os = "<</SMask<</S/Luminosity/G " .. format(pdfetcs.resfmt, on) .. ">>>>"
  on, new = update_pdfobjs(os)
  local key = add_extgs_resources(on,new)
  start_pdf_code()
  pdf_literalcode("/%s gs", key)
  if fd_stop then return "standalone" end
  return "start"
end

```

Transparency Group

```
pdfetcs.tr_group = { }
local function do_preobj_GRP (object, prescript)
  local grstate = prescript and prescript.gr_state
  if not grstate then return end
  local trgroup = pdfetcs.tr_group
  if grstate == "start" then
    trgroup.isolated, trgroup.knockout = false, false
    for _,v in ipairs(prescript.gr_type:explode",+") do
      trgroup[v] = true
    end
    local p = object.path
    trgroup.bbox = {
      math.min(p[1].x_coord, p[2].x_coord, p[3].x_coord, p[4].x_coord),
      math.min(p[1].y_coord, p[2].y_coord, p[3].y_coord, p[4].y_coord),
      math.max(p[1].x_coord, p[2].x_coord, p[3].x_coord, p[4].x_coord),
      math.max(p[1].y_coord, p[2].y_coord, p[3].y_coord, p[4].y_coord),
    }
```

```
    put2output[[\begingroup\setbox\mplibscratchbox\hbox\bgroup]]
  elseif grstate == "stop" then
    local llx,lly,urx,ury = tableunpack(trgroup.bbox)
    local grattr = format("/Group<</S/Transparency/I %s/K %s>>",trgroup.isolated,trgroup.knockout)
    local res = tableconcat(gather_resources(true))
    put2output(tableconcat{
      "\\egroup",
      format("\\wd\\mplibscratchbox %fbp", urx-llx),
      format("\\ht\\mplibscratchbox %fbp", ury-lly),
      "\\dp\\mplibscratchbox 0pt",
    })
    if pdfmode then
      put2output(tableconcat{
        "\\saveboxresource type 2 attr{/Type/XObject/Subtype/Form/FormType 1",
        format("/BBox[%f %f %f %f]", llx,lly,urx,ury),
        grattr, "} resources{", res, "}\\mplibscratchbox",
        [[\setbox\mplibscratchbox\hbox{\useboxresource\lastsavedboxresourceindex}]],
        [[\wd\mplibscratchbox 0pt\ht\mplibscratchbox 0pt\dp\mplibscratchbox 0pt]],
        [[\box\mplibscratchbox\endgroup]],
      })
    else
      trgroup.cnt = (trgroup.cnt or 0) + 1
      local objname = format("@mplibtrgr%s", trgroup.cnt)
      put2output(tableconcat{
        "\\special{pdf:bxobj ", objname, " bbox ", format("%f %f %f %f", llx,lly,urx,ury), "}",
        "\\unhbox\\mplibscratchbox",
        "\\special{pdf:put @resources <<", res, ">>}",
        "\\special{pdf:exobj <<", grattr, ">>}",
        "\\special{pdf:uxobj ", objname, "}\\endgroup",
      })
    end
  end
  return grstate
end

local function stop_special_effects(fade,opaq,over)
  if fade then -- fading
    stop_pdf_code()
  end
  if opaq then -- opacity
    pdf_literalcode(opaq)
  end
  if over then -- color
    put2output"\\special{pdf:ec}"
  end
end

```

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

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

```

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

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

local bend_tolerance = 131/65536

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

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

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

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

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

```
      pdf_literalcode("%f %f %f %f %f %f c",pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coord,one.y_coord )
    else
      pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
    end
  elseif #path == 1 then -- special case .. draw point
    local one = path[1]
    pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
  end
end

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

```

Finally, flush figures by inserting PDF literals.

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

```
        if urx < llx then
```

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

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

```
        else
```

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

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

The following 8 lines are part of btex...etex patch. Again, colors are processed at this stage.

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

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

```
                if prescript and prescript.postmplibverbtex then
                  figcontents.post[#figcontents.post+1] = prescript.postmplibverbtex
                end
              elseif objecttype == "text" then
                local ot = object.transform -- 3,4,5,6,1,2
                start_pdf_code()
                pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
                pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
                stop_pdf_code()
              elseif not trgroup and fading_ ~= "stop" then
```

```
2459              local evenodd, collect, both = false, false, false
2460              local postscript = object.postscript
2461              if not object.istext then
2462                if postscript == "evenodd" then
2463                  evenodd = true
2464                elseif postscript == "collect" then
2465                  collect = true
2466                elseif postscript == "both" then
2467                  both = true
2468                elseif postscript == "eoboth" then
2469                  evenodd = true
2470                  both    = true
2471                end
2472              end
2473              if collect then
2474                if not savedpath then
2475                  savedpath = { object.path or false }
2476                  savedhtap = { object.htap or false }
2477                else
2478                  savedpath[#savedpath+1] = object.path or false
2479                  savedhtap[#savedhtap+1] = object.htap or false
2480                end
2481              else
```

Removed from ConTeXt general: color stuff.

```
2482                local ml = object.miterlimit
2483                if ml and ml ~= miterlimit then
2484                  miterlimit = ml
2485                  pdf_literalcode("%f M",ml)
2486                end
2487                local lj = object.linejoin
2488                if lj and lj ~= linejoin then
2489                  linejoin = lj
2490                  pdf_literalcode("%i j",lj)
2491                end
2492                local lc = object.linecap
2493                if lc and lc ~= linecap then
2494                  linecap = lc
2495                  pdf_literalcode("%i J",lc)
2496                end
2497                local dl = object.dash
2498                if dl then
2499                  local d = format("[%s] %f d",tableconcat(dl.dashes or {}," "),dl.offset)
2500                  if d ~= dashed then
2501                    dashed = d
2502                    pdf_literalcode(dashed)
2503                  end
2504                elseif dashed then
2505                  pdf_literalcode("[] 0 d")
2506                  dashed = false
2507                end
```

Added : shading and pattern

```
2508                local shade_no = do_preobj_SH(object,prescript) -- shading
2509                local pattern_ = do_preobj_PAT(object,prescript) -- pattern
```

```
local path = object.path
local transformed, penwidth = false, 1
local open = path and path[1].left_type and path[#path].right_type
local pen = object.pen
if pen then
  if pen.type == 'elliptical' then
    transformed, penwidth = pen_characteristics(object) -- boolean, value
    pdf_literalcode("%f w",penwidth)
    if objecttype == 'fill' then
      objecttype = 'both'
    end
  else -- calculated by mplib itself
    objecttype = 'fill'
  end
end
if transformed then
  start_pdf_code()
end
if path then
  if savedpath then
    for i=1,#savedpath do
      local path = savedpath[i]
      if transformed then
        flushconcatpath(path,open)
      else
        flushnormalpath(path,open)
      end
    end
    savedpath = nil
  end
  if transformed then
    flushconcatpath(path,open)
  else
    flushnormalpath(path,open)
  end
```

Shading seems to conflict with these ops

```
  if not shade_no then -- conflict with shading
    if objecttype == "fill" then
      pdf_literalcode(evenodd and "h f*" or "h f")
    elseif objecttype == "outline" then
      if both then
        pdf_literalcode(evenodd and "h B*" or "h B")
      else
        pdf_literalcode(open and "S" or "h S")
      end
    elseif objecttype == "both" then
      pdf_literalcode(evenodd and "h B*" or "h B")
    end
  end
end
if transformed then
  stop_pdf_code()
end
local path = object.htap
```

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

Added to ConTeXt general: post-object colors and shading stuff. We should beware the `q ... Q` scope.

```
            if shade_no then -- shading
              pdf_literalcode("W n /MPlibSh%s sh Q",shade_no)
            end
          end
        end
        if fading_ == "start" then
          pdfetcs.fading.specialeffects = {fading_, tr_opaq, cr_over}
        elseif trgroup == "start" then
          pdfetcs.tr_group.specialeffects = {fading_, tr_opaq, cr_over}
        elseif fading_ == "stop" then
          local se = pdfetcs.fading.specialeffects
          if se then stop_special_effects(se[1], se[2], se[3]) end
        elseif trgroup == "stop" then
          local se = pdfetcs.tr_group.specialeffects
          if se then stop_special_effects(se[1], se[2], se[3]) end
        else
          stop_special_effects(fading_, tr_opaq, cr_over)
        end
        if fading_ or trgroup then -- extgs resetted
```

```
                miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
              end
            end
          end
          stop_pdf_code()
          pdf_stopfigure()
```

output collected materials to PDF, plus legacy verbatimtex code.

```
          for _,v in ipairs(figcontents) do
            if type(v) == "table" then
              texsprint"\\mplibtoPDF{"; texsprint(v[1], v[2]); texsprint"}"
            else
              texsprint(v)
            end
          end
          if #figcontents.post > 0 then texsprint(figcontents.post) end
          figcontents = { post = { } }
        end
      end
    end
  end
end

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

2.2 TEX package

First we need to load some packages.

```
\bgroup\expandafter\expandafter\expandafter\egroup
\expandafter\ifx\csname selectfont\endcsname\relax
  \input ltluatex
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luamplib}
    [2024/07/17 v2.34.0 mplib package for LuaTeX]
  \ifx\newluafunction\@undefined
  \input ltluatex
  \fi
\fi
```

Loading of lua code.

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

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

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

Set the format for metapost.

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

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

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

To make mplibcode typeset always in horizontal mode.

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

Catcode. We want to allow comment sign in mplibcode.

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

Make btex...etex box zero-metric.

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

Patterns

```
{\def\:{\global\let\mplibsptoken= } \: }
\protected\def\mppattern#1{%
  \begingroup
  \def\mplibpatternname{#1}%
  \mplibpatterngetnexttok
}
\def\mplibpatterngetnexttok{\futurelet\nexttok\mplibpatternbranch}
\def\mplibpatternskipspace{\afterassignment\mplibpatterngetnexttok\let\nexttok= }
\def\mplibpatternbranch{%
  \ifx [\nexttok
    \expandafter\mplibpatternopts
  \else
    \ifx\mplibsptoken\nexttok
      \expandafter\expandafter\expandafter\mplibpatternskipspace
    \else
      \let\mplibpatternoptions\empty
```

```
      \expandafter\expandafter\expandafter\mplibpatternmain
    \fi
  \fi
}
\def\mplibpatternopts[#1]{%
  \def\mplibpatternoptions{#1}%
  \mplibpatternmain
}
\def\mplibpatternmain{%
  \setbox\mplibscratchbox\hbox\bgroup\ignorespaces
}
\protected\def\endmppattern{%
  \egroup
  \directlua{ luamplib.registerpattern(
    \the\mplibscratchbox, '\mplibpatternname', {\mplibpatternoptions}
  )}%
  \endgroup
}
```

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

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

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

The Plain-specific stuff.

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

The LaTeX-specific part: a new environment.

```
  \newenvironment{mplibcode}[1][]{%
    \global\def\currentmpinstancename{#1}%
    \mplibtmptoks{}\ltxdomplibcode
```

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

User settings.

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

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

```
\ifcsname ver@luamplib.sty\endcsname
  \protected\def\everymplib{%
    \begingroup
    \mplibsetupcatcodes
    \mplibdoeverymplib
  }
```

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

Allow TEX dimen/color macros. Now `runscript` does the job, so the following lines are not needed for most cases.

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

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

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

```
  luamplib.numbersystem = t
}}
```

Settings for `.mp` cache files.

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

More user settings.

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

The followings are from ConTeXt general, mostly.

We use a dedicated scratchbox.

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

We encapsulate the literals.

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

Text items have a special handler.

```
\def\mplibtextext#1#2#3#4#5{%
  \begingroup
  \setbox\mplibscratchbox\hbox
    {\font\temp=#1 at #2bp%
     \temp
     #3}%
  \setbox\mplibscratchbox\hbox
    {\hskip#4 bp%
     \raise#5 bp%
     \box\mplibscratchbox}%
  \wd\mplibscratchbox0pt%
  \ht\mplibscratchbox0pt%
  \dp\mplibscratchbox0pt%
  \box\mplibscratchbox
  \endgroup
}
```

Input `luamplib.cfg` when it exists.

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

That's all folks!

3 The GNU GPL License v2

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

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

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

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

Preamble

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

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

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

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

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

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

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

Terms and Conditions For Copying, Distribution and Modification

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

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

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

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

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

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

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

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

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

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

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

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

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

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

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

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

No Warranty

12. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

13. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

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

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

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

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

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

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

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

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

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

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

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

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

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