The luamplib package

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

2025/03/20 v2.37.2

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` ⟨*string*⟩ is a more versatile macro equivalent to `TEX` ⟨*string*⟩ 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 § 1.1.
- 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

1.1.1 `\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.)

1.1.2 `\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}
```

1.1.3 `\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), shading (gradient colors) and transparency group are fully supported, and outlinetext is supported by our own alternative `mpliboutlinetext` (see below § 1.2). You can try other effects as well, though we did not fully tested their proper functioning.

transparency (`texdoc metafun` § 8.2) Transparency is so simple that you can apply it to an object, with *plain* format as well as *metafun*, just by appending `withprescript "tr_transparency=`⟨*number*⟩`"` to the sentence. ($0 \le \langle number \rangle \le 1$)

From v2.36, `withtransparency` is available with *plain* as well. See below § 1.2.

shading (`texdoc metafun` § 8.3) 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 a **color**, **xcolor** or **l3color**'s expression.

From v2.36, shading is available with *plain* format as well with extended functionality. See below § 1.2.

transparency group (`texdoc metafun` § 8.8) As for transparency group, the current *metafun* document 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.

Transparency group is available with *plain* format as well, with extended functionality. See below § 1.2.

1.1.4 `\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}`.

1.1.5 `\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`.

1.1.6 `\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` ⟨*string*⟩ 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{disable}` 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);
```

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

```
    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}
```

1.1.7 **`\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 (the text part) will be typeset with the current TeX font.

From v2.35, however, the redefinition of `infont` operator has been revised: when the character code of the text argument is less than 32 (control characters), or is equal to 35 (#), 36 ($), 37 (%), 38 (&), 92 (\), 94 (^), 95 (_), 123 ({), 125 (}), 126 (~) or 127 (DEL), the original `infont` operator will be used instead of `textext` operator so that the font part will be honored. Despite the revision, please take care of `char` operator in the text argument, as this might bring unpermitted characters into TeX.

1.1.8 **`\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.

1.1.9 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]{...}
```

1.1.10 \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
```

1.1.11 \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.

1.1.12 \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}
```

1.1.13 \mpcolor[...]{...}

With \mpcolor command, color names or expressions of color, xcolor and l3color module/packages can be used in the mplibcode environment (after withcolor operator). See the example above. The optional [...] denotes 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.

1.1.14 `\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.

1.1.15 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 nececcsary, 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.

1.1.16 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`.

1.1.17 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.

1.1.18 Tagged PDF

When tagpdf package is loaded and activated, `mplibcode` environment accepts additional options for tagged PDF. The code related to this functionality is currently in experimental stage, not guaranteeing backward compatibility. Like the LaTeX's `picture` environment, available optional keys are `tag`, `alt`, `actualtext`, `artifact`, `debug` and `correct-BBox` (texdoc `latex-lab-graphic`). Additionally, luamplib provides its own `text` key.

`tag=...` You can choose a tag name, default value being `Figure`. BBox info will be added automatically to the PDF unless the value is `text` or `false`. When the value is `false`, tagging is deactivated.

`debug` draws bounding box of the figure for checking, which you can correct by `correct-BBox` key with space-separated four dimen values.

`alt=...` sets an alternative text of the figure as given. This key is needed for ordinary METAPOST figures. You can give alternative text within METAPOST code as well: `VerbatimTeX "\mplibalttext{...}";`

`actualtext=...` starts a `Span` tag implicitly and sets an actual text as given. Horizontal mode is forced by `\noindent` command. BBox info will not be added. This key is intended for figures which can be represented by a character or a small sequence of characters. You can give actual text within METAPOST code as well: `VerbatimTeX "\mplibactualtext{...}";`

`artifact` starts an `artifact` MC (marked content). BBox info will not be added. This key is intended for decorative figures which have no semantic quality.

`text` starts an `artifact` MC and enables tagging on textext (the same as `btex ... etex`) boxes. Horizontal mode is forced by `\noindent` command. BBox info will not be added. This key is intended for figures made mostly of textext boxes. Inside text-keyed figures, reusing textext boxes is strongly discouraged.

These keys are provided also for `\mpfig` and `\usemplibgroup` (see below) commands.

```
\begin{mplibcode}[myInstanceName, alt=figure drawing a circle]
...
\end{mplibcode}

\mpfig[alt=figure drawing a square box]
...
\endmpfig

\usemplibgroup[alt=figure drawing a triangle]{...}

\mppattern{...}          % see below
  \mpfig[tag=false]      % do not tag this figure
  ...
  \endmpfig
\endmppattern
```

As for the instance name of `mplibcode` environment, `instance=...` or `instancename=...` is also allowed in addition to the raw instance name as shown above.

1.2 MetaPost

1.2.1 `mplibdimen ...`, `mplibcolor ...`

These are METAPOST interfaces for the TEX commands `\mpdim` and `\mpcolor` (see above). 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`.

1.2.2 `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.

1.2.3 `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 color, xcolor 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 with withshademethod from *metafun*. (But this limitation is now lifted by the introduction of withshadingmethod. See below.) Again, in DVI mode, unicode-math package is needed for math formula, as we cannot embolden type1 fonts in DVI mode.

1.2.4 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.

1.2.5 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 containing 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, with *plain* format as well, additionally declare withpostscript "evenodd" to the last path in the picture.

1.2.6 mpliboutlinetext (...)

From v2.31, a new METAPOST operator mpliboutlinetext is available, which mimicks *metafun*'s outlinetext. So the syntax is the same: 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.

1.2.7 \mppattern{...} ... \endmppattern, ... withpattern ..., withmppattern ...

TeX macros `\mppattern{`⟨*name*⟩`}` ... `\endmppattern` define a tiling pattern associated with the ⟨*name*⟩. METAPOST operator `withpattern`, the syntax being ⟨*path*⟩ | ⟨*textual picture*⟩ `withpattern` ⟨*string*⟩, will return a METAPOST picture which fills the given path or text with a tiling pattern of the ⟨*name*⟩ by replicating it horizontally and vertically. The *textual picture* here means any text typeset by TeX, mostly the result of the `btex` command (though technically this is not a true textual picture) or the `infont` operator.

`withmppattern` ⟨*string*⟩ is a command virtually the same as `withpattern`, but the former does not force the result of METAPOST picture. So users can use any drawing command suitable, such as `fill` or `filldraw` as well as `draw`.

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,
    draw (origin--(1,1))
      scaled 10
      withcolor 1/3[blue,white]
      ;
    draw (up--right)
      scaled 10
      withcolor 1/3[red,white]
      ;
  \endmpfig
\endmppattern                   % or \end{mppattern}

\mpfig
  draw fullcircle scaled 90
    withpostscript "collect"
    ;
  filldraw fullcircle scaled 200
    withmppattern "mypatt"
    withpen pencircle scaled 1
    withcolor \mpcolor{red!50!blue!50}
    withpostscript "evenodd"
    ;
```

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
bbox	*table* or *string*	llx, lly, urx, ury values*
matrix	*table* or *string*	xx, yx, xy, yy values* or MP transform code
resources	*string*	PDF resources if needed
colored or coloured	*boolean*	false for uncolored pattern. default: true

* in string type, numbers are separated by spaces

```
\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.

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}{pattnocolor}
  [
    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;
      filldraw pathpart item scaled 10
      if j < length mpliboutlinepic[i]:
          withpostscript "collect"
      else:
          withmppattern "pattnocolor"
          withpen pencircle scaled 1/2
          withcolor (i/4)[red,blue]         % paints the pattern
      fi;
    endfor
  endfor
  endfig;
\end{mplibcode}
```

A much simpler and efficient way to obtain a similar result (without colorful characters in this example) is to give a *textual picture* as the operand of `withpattern` or `withmppattern`:

```
  \begin{mplibcode}
    beginfig(2)
    picture pic;
    pic = mplibgraphictext "\bfseries\TeX"
            fakebold 1/2
            fillcolor 1/3[red,blue]           % paints the pattern
            drawcolor 2/3[red,blue]
            scaled 10 ;
    draw pic withmppattern "pattnocolor" ;
    endfig;
  \end{mplibcode}
```

1.2.8 ... **`withfademethod`** ...

This 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` from *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. Particularly, see the description below on the analogous macro `withgroupbbox`.

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
```

1.2.9 **... asgroup ...**

As said before, transparency group is available with *plain* as well as *metafun* format. The syntax is exactly the same: ⟨*picture*⟩ | ⟨*path*⟩ `asgroup` `""` | `"isolated"` | `"knockout"` | `"isolated,knockout"`, which will return a METAPOST picture. It is called *Transparency Group* because the objects contained in the group are composited to produce a single object, so that outer transparency effect, if any, will be applied to the group as a whole, not to the individual objects cumulatively.

The additional feature provided by luamplib is that you can reuse the group as many times as you want in the TEX code or in other METAPOST code chunks, with infinitesimal increase in the size of PDF file. For this functionality we provide TEX and METAPOST macros as follows:

`withgroupname` ⟨*string*⟩ associates a transparency group with the given name. When this is not appended to the sentence with `asgroup` operator, the default group name '`lastmplibgroup`' will be used.

`\usemplibgroup{`⟨*name*⟩`}` is a TEX command to reuse a transparency group of the name once used. Note that the position of the group will be origin-based: in other words, lower-left corner of the group will be shifted to the origin.

`usemplibgroup` ⟨*string*⟩ is a METAPOST command which will add a transparency group of the name to the `currentpicture`. Contrary to the TEX command just mentioned, the position of the group is the same as the original transparency group.

`withgroupbbox` (*pair*,*pair*) sets the bounding box of the transparency group, default value being `(llcorner p, urcorner p)`. This option might be needed especially when you draw with a thick pen a path that touches the boundary; you would probably want to append to the sentence '`withgroupbbox (bot lft llcorner p, top rt urcorner p)`', supposing that the pen was selected by the `pickup` command.

An example showing the difference between the TEX and METAPOST commands:

```
\mpfig
  draw image(
    fill fullcircle scaled 100 shifted 25right withcolor blue;
    fill fullcircle scaled 100 withcolor red ;
  ) asgroup ""
    withgroupname "mygroup";
  draw (left--right) scaled 10;
  draw (up--down) scaled 10;
\endmpfig

\noindent
```

Table 2: options for \mplibgroup

Key	Value Type	Explanation
asgroup	*string*	"", "isolated", "knockout", or "isolated,knockout"
bbox	*table* or *string*	llx, lly, urx, ury values*
matrix	*table* or *string*	xx, yx, xy, yy values* or MP transform code
resources	*string*	PDF resources if needed
		* in string type, numbers are separated by spaces

```
\clap{\vrule width 20pt height .25pt depth .25pt}%
\clap{\vrule width .5pt height 10pt depth 10pt}%
\usemplibgroup{mygroup}

\mpfig
  usemplibgroup "mygroup" rotated 15
    withtransparency (1, 0.5) ;
  draw (left--right) scaled 10;
  draw (up--down) scaled 10;
\endmpfig
```

Also note that normally the reused transparency groups are not affected by outer color commands. However, if you have made the original transparency group using `withoutcolor` command, colors will have effects on the uncolored objects in the group.

1.2.10 `\mplibgroup{...} ... \endmplibgroup`

These TEX macros are described here in this subsection, as they are deeply related to the asgroup operator. Users can define a transparency group or a normal *form XObject* with these macros from TEX side. The syntax is similar to the `\mppattern` command (see above). An example:

```
\mplibgroup{mygrx}                  % or \begin{mplibgroup}{mygrx}
  [                                 % options: see below
    asgroup="",
  ]
  \mpfig                            % or any other TeX code
    pickup pencircle scaled 10;
    draw (left--right) scaled 30 rotated 45 ;
    draw (left--right) scaled 30 rotated -45 ;
  \endmpfig
\endmplibgroup                      % or \end{mplibgroup}

\usemplibgroup{mygrx}

\mpfig
  usemplibgroup "mygrx" scaled 1.5
    withtransparency (1, 0.5) ;
\endmpfig
```

Availabe options, much fewer than those for `\mppattern`, are listed in Table 2. Again, the width/height/depth values of the mplibgroup will be written down into the log file.

When asgroup option, including empty string, is not given, a normal form XObject will be generated rather than a transparency group. Thus the individual objects, not the XObject as a whole, will be affected by outer transparency command.

As shown in the example, you can reuse the mplibgroup once defined using the TeX command `\usemplibgroup` or the METAPOST command `usemplibgroup`. The behavior of these commands is the same as that described above, excepting that mplibgroup made by TeX code (not by METAPOST code) respects original height and depth.

1.2.11 ... **`withtransparency`** ...

`withtransparency(`*number* | *string*, *number*`)` is provided for *plain* format as well. The first argument accepts a number or a name of alternative transparency methods (see `texdoc metafun` § 8.2 Figure 8.1). The second argument accepts a number denoting opacity.

```
fill fullcircle scaled 10
    withcolor red
    withtransparency (1, 0.5)          % or ("normal", 0.5)
    ;
```

1.2.12 ... **`withshadingmethod`** ...

The syntax is exactly the same as *metafun*'s new shading method (`texdoc metafun` § 8.3.3), except that the '`shade`' contained in each and every macro name has changed to '`shading`' in luamplib: for instance, while `withshademethod` is a macro name which only works with *metafun* format, the equivalent provided by luamplib, `withshadingmethod`, works with *plain* as well. Other differences to the *metafun*'s and some cautions are:

- *textual pictures* (pictures made by `btex` ... `etex`, `textext`, `maketext`, `mplibgraphictext`, `TEX`, `infont`, etc) as well as paths can have shading effect.

  ```
  draw btex \bfseries\TeX etex scaled 10
      withshadingmethod "linear"
      withshadingcolors (red,blue) ;
  ```

- When you give shading effect to a picture made by '`infont`' operator, the result of `withshadingvector` will be the same as that of `withshadingdirection`, as luamplib considers only the bounding box of the picture.

Macros provided by luamplib are:

⟨*path*⟩ | ⟨*textual picture*⟩ `withshadingmethod` ⟨*string*⟩ where ⟨*string*⟩ shall be `"linear"` or `"circular"`. This is the only 'must' item to get shading effect; all the macros below are optional.

`withshadingvector` ⟨*pair*⟩ Starting and ending points (as time value) on the path.

`withshadingdirection` ⟨*pair*⟩ Starting and ending points (as time value) on the bounding box. Default value: `(0,2)`

`withshadingorigin` ⟨*pair*⟩ The center of starting and ending circles. Default value: `center p`

`withshadingradius` ⟨*pair*⟩ Radii of starting and ending circles. This is no-op in linear mode. Default value: `(0, abs(center p - urcorner p))`

`withshadingfactor` ⟨*number*⟩ Multiplier of the radii. This is no-op in linear mode. Default value: `1.2` `withshadingcenter` ⟨*pair*⟩ Values for shifting starting center. For instance, `(0,0)` means that the center of starting circle is `center p`; `(1,1)` means `urcorner p`.

`withshadingtransform` ⟨*string*⟩ where ⟨*string*⟩ shall be `"yes"` (respect transform) or `"no"` (ignore transform). Default value: `"no"` for pictures made by `infont` operator; `"yes"` for all other cases.

`withshadingdomain` ⟨*pair*⟩ Limiting values of parametric variable that varies on the axis of color gradient. Default value: `(0,1)`

`withshadingstep (...)` for combined shading of more than two colors.

`withshadingfraction` ⟨*number*⟩ Fractional number of each shading step. Only meaningful with `withshadingstep`.

`withshadingcolors` (*color expr*, *color expr*) Starting and ending colors. Default value: `(white,black)`

1.2.13 **`mpliblength ...`**, **`mplibuclength ...`**

`mpliblength` ⟨*string*⟩ returns the number of unicode characters in the string. This is a unicode-aware version equivalent to the METAPOST primitive `length`, but accepts only a string-type argument. For instance, `mpliblength "abçdéf"` returns `6`, not `8`.

On the other hand, `mplibuclength` ⟨*string*⟩ returns the number of unicode grapheme clusters in the string. For instance, `mplibuclength "Äpfel"`, where `Ä` is encoded using two codepoints (U+0041 and U+0308), returns `5`, not `6` or `7`. This operator requires lua-uni-algos package.

1.2.14 **`mplibsubstring ... of ...`**, **`mplibucsubstring ... of ...`**

`mplibsubstring` ⟨*pair*⟩ `of` ⟨*string*⟩ is a unicode-aware version equivalent to the METAPOST's `substring ... of ...` primitive. The syntax is the same as the latter, but the string is indexed by unicode characters. For instance, `mplibsubstring (2,5) of "abçdéf"` returns `"çdé"`, and `mplibsubstring (5,2) of "abçdéf"` returns `"édç"`.

On the other hand, `mplibucsubstring` ⟨*pair*⟩ `of` ⟨*string*⟩ returns the part of the string indexed by unicode grapheme clusters. For instance, `mplibucsubstring (0,1) of "Äpfel"`, where `Ä` is encoded using two codepoints (U+0041 and U+0308), returns `"Ä"`, not `"A"`. This operator requires lua-uni-algos package.

1.3 Lua

1.3.1 **`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.

Table 3: 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

1.3.2 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]
  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
}
```

1.3.3 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 3, can have effects on the process of process_mplibcode.

2 Implementation

2.1 Lua module

```

luatexbase.provides_module {
  name          = "luamplib",
  version       = "2.37.2",
  date          = "2025/03/20",
  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 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 = lfsattributes(kpse.find_file"luamplib.lua", "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 .mp files 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 or other Lua functions. And we provide numbersystem option since v2.4. 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, maketext)
  if str then
    if not maketext then str = str:gsub("\r.-$","") end
    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("\\luamplibtagtextbegin{%i}%s\\setbox%i\\hbox{%s}\\luamplibtagtextend", tex_box_id, global,
    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("%%.-$",  "")
            :gsub("%zPerCent%z", "\\%%")
            :gsub("\r.-$",  "")
            :gsub("%s+", " ")
end
luamplib.legacyverbatimtex = true
function luamplib.maketext (str, what)
  if str and str ~= "" then
    str = protecttexcontents(str)
    if what == 1 then
      if not str:find("\\documentclass"..name_e) and
         not str:find("\\begin%s*{document}") and
         not str:find("\\documentstyle"..name_e) and
         not str:find("\\usepackage"..name_e) then
        if luamplib.legacyverbatimtex then
          if luamplib.in_the_fig then
            return process_verbatimtex_infig(str)
          else
            return process_verbatimtex_prefig(str)
          end
        else
          return process_verbatimtex_text(str)
        end
      end
    else
      return process_tex_text(str, true) -- bool is for 'char13'
    end
  end
  return ""
end

```

luamplib's METAPOST color operators

```
local function colorsplit (res)
  local t, tt = { }, res:gsub("[%[%]]","",2):explode()
  local be = tt[1]:find"^%d" and 1 or 2
  for i=be, #tt do
    if not tonumber(tt[i]) then break end
    t[#t+1] = tt[i]
  end
  return t
end

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

```
\DocumentMetadata{ }
\documentclass{article}
\usepackage{luamplib}
\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 xscaled \mpdim\textwidth yscaled 1cm
       withshadingmethod "linear"
       withshadingvector (0,1)
       withshadingstep (
           withshadingfraction .5
           withshadingcolors ("spotB","spotC")
       )
       withshadingstep (
           withshadingfraction 1
           withshadingcolors ("spotC","spotD")
       )
  ;
endfig;
\end{mplibcode}
\end{document}
```

another one: user-defined DeviceN colorspace

```
\DocumentMetadata{ }
\documentclass{article}
\usepackage{luamplib}
\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
```

```
    withshadingmethod "linear"
    withshadingcolors ("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

```

Remove trailing zeros for smaller PDF

```
local decimals = "%.%d+"
local function rmzeros(str) return str:gsub("%.?0+$","") end

```

luamplib's mplibgraphictext operator

```
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") :gsub(decimals,rmzeros)
  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 running = -1073741824
  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 key = format("%s:%s",f,fakebold)
      local i = emboldenfonts[key]
      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(';$','')
          name = format('%s;embolden=%s;',name,fakebold)
          _, i = fonts.constructors.readanddefine(name,ft.size)
        end
        emboldenfonts[key] = i
      end
      curr.font = i
    end
    ::skip_type1::
    curr = node.getnext(curr)
  end
  return head
end
local function graphictextcolor (col, filldraw)
  if col:find"^[%d%.:]+$" then
    col = col:explode":"
    for i=1,#col do
      col[i] = format("%.3f", col[i])
    end
    if pdfmode then
      local op = #col == 4 and "k" or #col == 3 and "rg" or "g"
      col[#col+1] = filldraw == "fill" and op or op:upper()
      return tableconcat(col," ")
    end
    return format("[%s]", tableconcat(col," "))
  end
  col = process_color(col):match'"mpliboverridecolor=(.+)"'
  if pdfmode then
    local t, tt = col:explode(), { }
    local b = filldraw == "fill" and 1 or #t/2+1
    local e = b == 1 and #t/2 or #t
    for i=b,e do
      tt[#tt+1] = t[i]
    end
    return tableconcat(tt," ")
  end
  return col:gsub("^.- ","")
end
luamplib.graphictext = function (text, fakebold, fc, dc)
  local fmt = process_tex_text(text):sub(1,-2)
  local id = tonumber(fmt:match"mplibtexboxid=(%d+):")
  emboldenfonts.width = nil
  local box = texgetbox(id)
  box.head = embolden(box, box.head, fakebold)
  local fill = graphictextcolor(fc,"fill")
  local draw = graphictextcolor(dc,"draw")
  local bc = pdfmode and "" or "pdf:bc "
  return format('%s withprescript "mpliboverridecolor=%s%s %s"', fmt, bc, fill, draw)
end

```

luamplib's mplibglyph operator

```
local function mperr (str)
  return format("hide(errmessage %q)", str)
end
local function getangle (a,b,c)
  local r = math.deg(math.atan(c.y-b.y, c.x-b.x) - math.atan(b.y-a.y, b.x-a.x))
  if r > 180 then
    r = r - 360
  elseif r < -180 then
    r = r + 360
  end
  return r
end
local function turning (t)
  local r, n = 0, #t
  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 = math.floor((vwidth-diff) / hd)
              dy = dy + (b2u and diff or -diff)
            else
              n = math.floor(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
            break
          end
        end
      end
      local image
      if ft.format == "opentype" or ft.format == "truetype" then
        image = luamplib.glyph(curr.font, gid)
      else
        local name, scale = ft.name, 1
        local vf = font.read_vf(name, ft.size)
        if vf and vf.characters[gid] then
          local cmds = vf.characters[gid].commands or {}
          for _,v in ipairs(cmds) do
            if v[1] == "char" then
              gid = v[2]
            elseif v[1] == "font" and vf.fonts[v[2]] then
              name  = vf.fonts[v[2]].name
              scale = vf.fonts[v[2]].size / ft.size
            end
          end
        end
        image = format("glyph %s of %q scaled %f", gid, name, scale)
      end
      res[#res+1] = format("mpliboutlinepic[%i]:=%s xscaled %f yscaled %f slanted %f %s shifted (%f,%f);",
                           #res+1, image, xscale, yscale, slant, vertical, xpos, ypos)
      dx = dx + (r2l and 0 or curr.width/factor*expand)
    elseif curr.replace then
      local width = node.dimensions(curr.replace)/factor
      dx = dx - (r2l and width or 0)
      res = outline_horz(res, box, curr.replace, xshift+dx, yshift, width)
      dx = dx + (r2l and 0 or width)
    elseif curr.id == node.id"rule" then
      local wd, ht, dp = getrulemetric(box, curr, true)
      if wd ~= 0 then
        local hd = ht + dp
        dx = dx - (r2l and wd or 0)
        if hd ~= 0 and curr.subtype == 0 then
          res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
        end
        dx = dx + (r2l and 0 or wd)
      end
    elseif curr.id == node.id"glue" then
      local width = node.effective_glue(curr, box)/factor
      dx = dx - (r2l and width or 0)
      if curr.leader then
        local curr, kind = curr.leader, curr.subtype
        if curr.id == node.id"rule" then
          local wd, ht, dp = getrulemetric(box, curr, true)
          local hd = ht + dp
          if hd ~= 0 then
            wd = width
            if wd ~= 0 and curr.subtype == 0 then
              res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
            end
          end
        elseif curr.head then
          local wd = curr.width/factor
          if wd <= width then
            local dx = r2l and dx+width or dx
            local n, ix = 0, 0
            if kind == 100 or kind == 103 then -- todo: gleaders
              local adx = abs(dx-dirs[1].dx)
              local ndx = math.ceil(adx / wd) * wd
              local diff = ndx - adx
              n = math.floor((width-diff) / wd)
              dx = dx + (r2l and -diff-wd or diff)
            else
              n = math.floor(width / wd)
              if kind == 101 then
                local side = width % wd /2
                dx = dx + (r2l and -side-wd or side)
              elseif kind == 102 then
                ix = width % wd / (n+1)
```

```
                dx = dx + (r2l and -ix-wd or ix)
              end
            end
            wd = r2l and -wd or wd
            ix = r2l and -ix or ix
            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+dx, yshift-curr.shift/factor)
              dx = dx + wd + ix
            end
          end
        end
      end
      dx = dx + (r2l and 0 or width)
    elseif curr.id == node.id"kern" then
      dx = dx + curr.kern/factor * (r2l and -1 or 1)
    elseif curr.id == node.id"math" then
      dx = dx + curr.surround/factor * (r2l and -1 or 1)
    elseif curr.id == node.id"vlist" then
      dx = dx - (r2l and curr.width/factor or 0)
      res = outline_vert(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
      dx = dx + (r2l and 0 or curr.width/factor)
    elseif curr.id == node.id"hlist" then
      dx = dx - (r2l and curr.width/factor or 0)
      res = outline_horz(res, curr, curr.head, xshift+dx, yshift-curr.shift/factor)
      dx = dx + (r2l and 0 or curr.width/factor)
    end
    curr = node.getnext(curr)
  end
  return res
end
function luamplib.outlinetext (text)
  local fmt = process_tex_text(text)
  local id  = tonumber(fmt:match"mplibtexboxid=(%d+):")
  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

```

lua functions for `mplib(uc)substring ... of ...`

```
function luamplib.getunicodegraphemes (s)
  local t = { }
  local graphemes = require'lua-uni-graphemes'
  for _, _, c in graphemes.graphemes(s) do
    table.insert(t, c)
  end
  return t
end
function luamplib.unicodesubstring (s,b,e,grph)
  local tt, t, step = { }
  if grph then
    t = luamplib.getunicodegraphemes(s)
  else
```

```
    t = { }
    for _, c in utf8.codes(s) do
      table.insert(t, utf8.char(c))
    end
  end
  if b <= e then
    b, step = b+1, 1
  else
    e, step = e+1, -1
  end
  for i = b, e, step do
    table.insert(tt, t[i])
  end
  s = table.concat(tt):gsub('"','"&ditto&"')
  return string.format('"%s"', s)
end

```

Our METAPOST preambles

```
luamplib.preambles = {
  mplibcode = [[
texscriptmode := 2;
def rawtextext primary t = runscript("luamplibtext{"&t&"}") enddef;
def mplibcolor primary t = runscript("luamplibcolor{"&t&"}") enddef;
def mplibdimen primary t = runscript("luamplibdimen{"&t&"}") enddef;
def VerbatimTeX primary 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@# primary t = rawtextext (t) enddef;
  def message expr t =
    if string t: runscript("mp.report[=["&t&"]=]") else: errmessage "Not a string" fi
  enddef;
  def withtransparency (expr a, t) =
    withprescript "tr_alternative=" & if numeric a: decimal fi a
    withprescript "tr_transparency=" & decimal t
  enddef;
  vardef ddecimal primary p =
    decimal xpart p & " " & decimal ypart p
  enddef;
  vardef boundingbox primary p =
    if (path p) or (picture p) :
      llcorner p -- lrcorner p -- urcorner p -- ulcorner p
    else :
      origin
    fi -- cycle
  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 ;
def withmppattern primary p =
  withprescript "mplibpattern=" & if numeric p: decimal fi p
```

```
enddef;
primarydef t withpattern p =
  image(
    if cycle t:
      fill
    else:
      draw
    fi
    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 -1/4) & ":" &
      decimal (ypart llcorner p -1/4) & ":" &
      decimal (xpart urcorner p +1/4) & ":" &
      decimal (ypart urcorner p +1/4)
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;
primarydef p asgroup s =
  image(
    draw center p
      withprescript "mplibgroupbbox=" &
        decimal (xpart llcorner p -1/4) & ":" &
        decimal (ypart llcorner p -1/4) & ":" &
        decimal (xpart urcorner p +1/4) & ":" &
        decimal (ypart urcorner p +1/4)
      withprescript "gr_state=start"
      withprescript "gr_type=" & s;
    draw p;
    draw center p withprescript "gr_state=stop";
  )
enddef;
def withgroupbbox (expr a,b) =
  withprescript "mplibgroupbbox=" &
    decimal xpart a & ":" &
    decimal ypart a & ":" &
    decimal xpart b & ":" &
    decimal ypart b
enddef;
def withgroupname expr s =
  withprescript "mplibgroupname=" & s
enddef;
def usemplibgroup primary s =
  draw maketext("\csname luamplib.group." & s & "\endcsname")
    shifted runscript("return luamplib.trgroupshifts['" & s & "']")
enddef;
path    mplib_shade_path ;
numeric mplib_shade_step ; mplib_shade_step := 0 ;
numeric mplib_shade_fx, mplib_shade_fy ;
numeric mplib_shade_lx, mplib_shade_ly ;
numeric mplib_shade_nx, mplib_shade_ny ;
numeric mplib_shade_dx, mplib_shade_dy ;
numeric mplib_shade_tx, mplib_shade_ty ;
primarydef p withshadingmethod m =
  p
  if picture p :
    withprescript "sh_operand_type=picture"
    if textual p:
      withprescript "sh_transform=no"
      mplib_with_shade_method (boundingbox p, m)
    else:
      withprescript "sh_transform=yes"
      mplib_with_shade_method (pathpart p, m)
    fi
  else :
    withprescript "sh_transform=yes"
    mplib_with_shade_method (p, m)
  fi
enddef;
def mplib_with_shade_method (expr p, m) =
  hide(mplib_with_shade_method_analyze(p))
  withprescript "sh_domain=0 1"
  withprescript "sh_color=into"
  withprescript "sh_color_a=" & colordecimals white
  withprescript "sh_color_b=" & colordecimals black
  withprescript "sh_first=" & ddecimal point 0 of p
  withprescript "sh_set_x=" & ddecimal (mplib_shade_nx,mplib_shade_lx)
  withprescript "sh_set_y=" & ddecimal (mplib_shade_ny,mplib_shade_ly)
  if m = "linear" :
    withprescript "sh_type=linear"
    withprescript "sh_factor=1"
    withprescript "sh_center_a=" & ddecimal llcorner p
    withprescript "sh_center_b=" & ddecimal urcorner p
  else :
    withprescript "sh_type=circular"
    withprescript "sh_factor=1.2"
    withprescript "sh_center_a=" & ddecimal center p
    withprescript "sh_center_b=" & ddecimal center p
    withprescript "sh_radius_a=" & decimal 0
    withprescript "sh_radius_b=" & decimal mplib_max_radius(p)
  fi
enddef;
def mplib_with_shade_method_analyze(expr p) =
  mplib_shade_path := p ;
  mplib_shade_step := 1 ;
  mplib_shade_fx   := xpart point 0 of p ;
  mplib_shade_fy   := ypart point 0 of p ;
  mplib_shade_lx   := mplib_shade_fx ;
  mplib_shade_ly   := mplib_shade_fy ;
  mplib_shade_nx   := 0 ;
  mplib_shade_ny   := 0 ;
  mplib_shade_dx   := abs(mplib_shade_fx - mplib_shade_lx) ;
  mplib_shade_dy   := abs(mplib_shade_fy - mplib_shade_ly) ;
  for i=1 upto length(p) :
    mplib_shade_tx := abs(mplib_shade_fx - xpart point i of p) ;
    mplib_shade_ty := abs(mplib_shade_fy - ypart point i of p) ;
    if mplib_shade_tx > mplib_shade_dx :
      mplib_shade_nx := i + 1 ;
      mplib_shade_lx := xpart point i of p ;
      mplib_shade_dx := mplib_shade_tx ;
    fi ;
    if mplib_shade_ty > mplib_shade_dy :
      mplib_shade_ny := i + 1 ;
      mplib_shade_ly := ypart point i of p ;
      mplib_shade_dy := mplib_shade_ty ;
    fi ;
  endfor ;
```

```
enddef;
vardef mplib_max_radius(expr p) =
  max (
    (xpart center   p - xpart llcorner p) ++ (ypart center   p - ypart llcorner p),
    (xpart center   p - xpart ulcorner p) ++ (ypart ulcorner p - ypart center   p),
    (xpart lrcorner p - xpart center   p) ++ (ypart center   p - ypart lrcorner p),
    (xpart urcorner p - xpart center   p) ++ (ypart urcorner p - ypart center   p)
  )
enddef;
def withshadingstep (text t) =
  hide(mplib_shade_step := mplib_shade_step + 1 ;)
  withprescript "sh_step=" & decimal mplib_shade_step
  t
enddef;
def withshadingradius expr a =
  withprescript "sh_radius_a=" & decimal (xpart a)
  withprescript "sh_radius_b=" & decimal (ypart a)
enddef;
def withshadingorigin expr a =
  withprescript "sh_center_a=" & ddecimal a
  withprescript "sh_center_b=" & ddecimal a
enddef;
def withshadingvector expr a =
  withprescript "sh_center_a=" & ddecimal (point xpart a of mplib_shade_path)
  withprescript "sh_center_b=" & ddecimal (point ypart a of mplib_shade_path)
enddef;
def withshadingdirection expr a =
  withprescript "sh_center_a=" & ddecimal (point xpart a of boundingbox(mplib_shade_path))
  withprescript "sh_center_b=" & ddecimal (point ypart a of boundingbox(mplib_shade_path))
enddef;
def withshadingtransform expr a =
  withprescript "sh_transform=" & a
enddef;
def withshadingcenter expr a =
  withprescript "sh_center_a=" & ddecimal (
    center mplib_shade_path shifted (
      xpart a * xpart (lrcorner mplib_shade_path - llcorner mplib_shade_path)/2,
      ypart a * ypart (urcorner mplib_shade_path - lrcorner mplib_shade_path)/2
    )
  )
enddef;
def withshadingdomain expr d =
  withprescript "sh_domain=" & ddecimal d
enddef;
def withshadingfactor expr f =
  withprescript "sh_factor=" & decimal f
enddef;
def withshadingfraction expr a =
  if mplib_shade_step > 0 :
    withprescript "sh_fraction_" & decimal mplib_shade_step & "=" & decimal a
  fi
enddef;
def withshadingcolors (expr a, b) =
  if mplib_shade_step > 0 :
    withprescript "sh_color=into"
    withprescript "sh_color_a_" & decimal mplib_shade_step & "=" & colordecimals a
    withprescript "sh_color_b_" & decimal mplib_shade_step & "=" & colordecimals b
  else :
    withprescript "sh_color=into"
    withprescript "sh_color_a=" & colordecimals a
    withprescript "sh_color_b=" & colordecimals b
  fi
enddef;
def mpliblength primary t =
  runscript("return utf8.len[===[" & t & "]===]")
enddef;
def mplibsubstring expr p of t =
  runscript("return luamplib.unicodesubstring([===[" & t & "]===],"
    & decimal xpart p & ","
    & decimal ypart p & ")")
enddef;
def mplibuclength primary t =
  runscript("return #luamplib.getunicodegraphemes[===[" & t & "]===]")
enddef;
def mplibucsubstring expr p of t =
  runscript("return luamplib.unicodesubstring([===[" & t & "]===],"
    & decimal xpart p & ","
    & decimal ypart p & ",true)")
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 = [[
let luampliboriginalinfont = infont;
primarydef s infont f =
  if   (s < char 32)
    or (s = char 35) % #
    or (s = char 36) % $
    or (s = char 37) % %
    or (s = char 38) % &
    or (s = char 92) % \
    or (s = char 94) % ^
    or (s = char 95) % _
    or (s = char 123) % {
    or (s = char 125) % }
    or (s = char 126) % ~
    or (s = char 127) :
    s luampliboriginalinfont f
  else :
    rawtextext(s)
  fi
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 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
      t[k] = v
    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 (...)
  put2output{ -2, (format(...) :gsub(decimals,rmzeros)) }
end
local start_pdf_code = pdfmode
  and function() pdf_literalcode"q" end
  or  function() put2output"\\special{pdf:bcontent}" end
local stop_pdf_code = pdfmode
  and function() pdf_literalcode"Q" end
  or  function() put2output"\\special{pdf:econtent}" 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:explode":"
  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 = key and pdfobjs[key]
  if on then
    return on,false
  end
  if pdfmode then
    if stream then
      on = pdf.immediateobj("stream",stream,os)
    elseif os then
      on = pdf.immediateobj(os)
    else
      on = pdf.reserveobj()
    end
  else
    on = pdfetcs.cnt or 1
    if stream then
      texsprint(format("\\special{pdf:stream @mplibpdfobj%s (%s) <<%s>>}",on,stream,os))
    elseif os then
      texsprint(format("\\special{pdf:obj @mplibpdfobj%s %s}",on,os))
    else
      texsprint(format("\\special{pdf:obj @mplibpdfobj%s <<>>}",on))
    end
    pdfetcs.cnt = on + 1
  end
  if key then
    pdfobjs[key] = on
  end
  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 {
    "\\luamplibatfirstshipout{",
    "\\special{pdf:obj @MPlibTr<<>>}",
    "\\special{pdf:obj @MPlibSh<<>>}",
    "\\special{pdf:obj @MPlibCS<<>>}",
    "\\special{pdf:obj @MPlibPt<<>>}}",
  }
  pdfetcs.resadded = { }
  pdfetcs.fallback_update_resources = function (name,res,obj)
    texsprint{"\\special{pdf:put ", obj, " <<", res, ">>}"}
    if not pdfetcs.resadded[name] then
      texsprint{"\\luamplibateveryshipout{\\special{pdf:put @resources <</", name, " ", obj, ">>}}"}
      pdfetcs.resadded[name] = obj
    end
  end
end

```

Transparency

```
local transparancy_modes = { [0] = "Normal",
  "Normal",       "Multiply",     "Screen",       "Overlay",
  "SoftLight",    "HardLight",    "ColorDodge",   "ColorBurn",
  "Darken",       "Lighten",      "Difference",   "Exclusion",
  "Hue",          "Saturation",   "Color",        "Luminosity",
  "Compatible",
  normal     = "Normal",     multiply   = "Multiply",   screen    = "Screen",
  overlay    = "Overlay",    softlight  = "SoftLight",  hardlight = "HardLight",
  colordodge = "ColorDodge", colorburn  = "ColorBurn",  darken    = "Darken",
  lighten    = "Lighten",    difference = "Difference", exclusion = "Exclusion",
  hue        = "Hue",        saturation = "Saturation", color     = "Color",
  luminosity = "Luminosity", compatible = "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 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,"@MPlibTr")
      end
    end
  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:lower()]
    if not mode then
      mode = prescript.tr_alternative
      warn("unsupported blend mode: '%s'", mode)
    end
    opaq = format("%.3f", opaq) :gsub(decimals,rmzeros)
    for i,v in ipairs{ {mode,opaq},{"Normal",1} } do
```

```
      os = format("<</BM/%s/ca %s/CA %s/AIS false>>",v[1],v[2],v[2])
      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)
  for _,v in ipairs{ca,cb} do
    for i,vv in ipairs(v) do
      for ii,vvv in ipairs(vv) do
        v[i][ii] = tonumber(vvv) and format("%.3f",vvv) or vvv
      end
    end
  end
  local fun2fmt,os = "<</FunctionType 2/Domain[%s]/C0[%s]/C1[%s]/N 1>>"
  if steps > 1 then
    local list,bounds,encode = { },{ },{ }
    for i=1,steps do
      if i < steps then
        bounds[i] = format("%.3f", fractions[i] or 1)
      end
      encode[2*i-1] = 0
      encode[2*i]   = 1
      os = fun2fmt:format(domain,tableconcat(ca[i],' '),tableconcat(cb[i],' '))
        :gsub(decimals,rmzeros)
      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),
    } :gsub(decimals,rmzeros)
  else
    os = fun2fmt:format(domain,tableconcat(ca[1],' '),tableconcat(cb[1],' '))
      :gsub(decimals,rmzeros)
  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>>",
  } :gsub(decimals,rmzeros)
  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)
      pdfetcs.fallback_update_resources("Shading",res,"@MPlibSh")
    end
  end
  return on
end
local function color_normalize(ca,cb)
  if #cb == 1 then
    if #ca == 4 then
      cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
    else -- #ca = 3
      cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
    end
  elseif #cb == 3 then -- #ca == 4
    cb[1], cb[2], cb[3], cb[4] = 1-cb[1], 1-cb[2], 1-cb[3], 0
  end
end
pdfetcs.clrspcs = 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"):explode()
    local centerb = (prescript.sh_center_b or "0 0"):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"):explode()
      local setx  = (prescript.sh_set_x or "0 0"):explode()
      local sety  = (prescript.sh_set_y or "0 0"):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"):explode":" }
    cb = { (prescript.sh_color_b_1 or prescript.sh_color_b or "1"):explode":" }
    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"):explode":"
        cb[i] = (prescript[format("sh_color_b_%i",i)] or "1"):explode":"
      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
  end
  return shade_no
end

```

Shading Patterns: much similar to the metafun's shade, but we can apply shading to textual pictures as well as paths.

```
if not pdfmode then
  pdfetcs.patternresources = {}
end
local function add_pattern_resources (key, val)
  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, "}" }
    else
      pdfetcs.fallback_update_resources("Pattern",res,"@MPlibPt")
      if not pdfmode then
        tableinsert(pdfetcs.patternresources, res) -- for gather_resources()
      end
    end
  end
end
function luamplib.dolatelua (on, os)
  local h, v = pdf.getpos()
  h = format("%f", h/factor) :gsub(decimals,rmzeros)
  v = format("%f", v/factor) :gsub(decimals,rmzeros)
  if pdfmode then
    pdf.obj(on, format("<<%s/Matrix[1 0 0 1 %s %s]>>", os, h, v))
    pdf.refobj(on)
  else
    local shift = os:explode()
    if tonumber(h) ~= tonumber(shift[1]) or tonumber(v) ~= tonumber(shift[2]) then
      warn([[Add 'withprescript "sh_matrixshift=%s %s"' to the picture shading]], h, v)
    end
  end
end
local function do_preobj_shading (object, prescript)
  if not prescript or not prescript.sh_operand_type then return end
  local on = do_preobj_SH(object, prescript)
  local os = format("/PatternType 2/Shading %s", format(pdfetcs.resfmt, on))
  on = update_pdfobjs()
  if pdfmode then
    put2output(tableconcat{ "\\latelua{ luamplib.dolatelua(",on,",[[",os,"]]) }" })
  else
```

Why @xpos @ypos do not work properly???
Anyway, this seems to be needed for proper functioning:

```
\pagewidth=\paperwidth
\pageheight=\paperheight
\special{papersize=\the\paperwidth,\the\paperheight}
```

```
    if is_defined"RecordProperties" then
      put2output(tableconcat{
        "\\csname tex_savepos:D\\endcsname\\RecordProperties{luamplib/getpos/",on,"}{xpos,ypos}\z
        \\special{pdf:put @mplibpdfobj",on," <<",os,"/Matrix[1 0 0 1 \z
        \\csname dim_to_decimal_in_bp:n\\endcsname{\\RefProperty{luamplib/getpos/",on,"}{xpos}sp} \z
        \\csname dim_to_decimal_in_bp:n\\endcsname{\\RefProperty{luamplib/getpos/",on,"}{ypos}sp}\z
        ]>>}"
      })
    else
      local shift = prescript.sh_matrixshift or "0 0"
      texsprint{ "\\special{pdf:put @mplibpdfobj",on," <<",os,"/Matrix[1 0 0 1 ",shift,"]>>}" }
      put2output(tableconcat{ "\\latelua{ luamplib.dolatelua(",on,",[[",shift,"]]) }" })
    end
  end
  local key, val = format("MPlibPt%s", on), format(pdfetcs.resfmt, on)
  add_pattern_resources(key,val)
  pdf_literalcode("/Pattern cs/%s scn", key)
```

To avoid possible double execution, once by Pattern gs, once by Sh operator.

```
  prescript.sh_type = nil
end

```

Tiling Patterns

```
pdfetcs.patterns = { }
local function gather_resources (optres)
  local t, do_pattern = { }, not optres
  local names = {"ExtGState","ColorSpace","Shading"}
  if do_pattern then
    names[#names+1] = "Pattern"
  end
  if pdfmode then
    if pdfmanagement then
      for _,v in ipairs(names) do
        if ltx.__pdf.Page.Resources[v] then
          t[#t+1] = format("/%s %s 0 R", v, ltx.pdf.object_id("__pdf/Page/Resources/"..v))
        end
      end
    else
      local res = pdfetcs.getpageres() or ""
      run_tex_code[[\mplibtmptoks\expandafter{\the\pdfvariable pageresources}]]
      res = res .. texgettoks'mplibtmptoks'
      if do_pattern then return res end
      res = res:explode"/+"
      for _,v in ipairs(res) do
        v = v:match"^%s*(.-)%s*$"
        if not v:find"Pattern" and not optres:find(v) then
          t[#t+1] = "/" .. v
        end
      end
    end
  else
    if pdfmanagement then
      for _,v in ipairs(names) do
        run_tex_code ({
          "\\mplibtmptoks\\expanded{{",
          "\\pdfdict_if_empty:nF{g__pdf_Core/Page/Resources/", v, "}",
          "{/", v, " \\pdf_object_ref:n{__pdf/Page/Resources/", v, "}}}}",
        },ccexplat)
        t[#t+1] = texgettoks'mplibtmptoks'
      end
    elseif is_defined(pdfetcs.pgfextgs) then
      run_tex_code ({
        "\\mplibtmptoks\\expanded{{",
        "\\ifpgf@sys@pdf@extgs@exists /ExtGState @pgfextgs\\fi",
        "\\ifpgf@sys@pdf@colorspaces@exists /ColorSpace @pgfcolorspaces\\fi",
        do_pattern and "\\ifpgf@sys@pdf@patterns@exists /Pattern @pgfpatterns \\fi" or "",
        "}}",
      }, catat11)
      t[#t+1] = texgettoks'mplibtmptoks'
      if pdfetcs.resadded.Shading then
        t[#t+1] = format("/Shading %s", pdfetcs.resadded.Shading)
      end
    else
      for _,v in ipairs(names) do
        local vv = pdfetcs.resadded[v]
        if vv then
          t[#t+1] = format("/%s %s", v, vv)
        end
      end
    end
  end
  if do_pattern then return tableconcat(t) end
  -- get pattern resources
  local mytoks
  if pdfmanagement then
    run_tex_code ({
      "\\mplibtmptoks\\expanded{{",
      "\\pdfdict_if_empty:nF{g__pdf_Core/Page/Resources/Pattern}",
      "{\\pdfdict_use:n{g__pdf_Core/Page/Resources/Pattern}}", "}}",
    },ccexplat)
    mytoks = texgettoks"mplibtmptoks"
    if not pdfmode then
      mytoks = mytoks:gsub("\\str_convert_pdfname:n%s*{(.-)}","%1") -- why not expanded?
    end
  elseif is_defined(pdfetcs.pgfextgs) then
    if pdfmode then
      mytoks = get_macro"pgf@sys@pgf@resource@list@patterns"
    else
      local tt, abc = {}, get_macro"pgfutil@abc" or ""
      for v in abc:gmatch"@pgfpatterns%s*<<(.-)>>" do
        tt[#tt+1] = v
      end
      mytoks = tableconcat(tt)
    end
  else
    local tt = pdfmode and pdfetcs.Pattern_res or pdfetcs.patternresources
    mytoks = tt and tableconcat(tt)
  end
  if mytoks and mytoks ~= "" then
    t[#t+1] = format("/Pattern<<%s>>",mytoks)
  end
  return tableconcat(t)
end
function luamplib.registerpattern ( boxid, name, opts )
  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 pattern '%s': %s 0", name, format("%s %s",wd, hd):gsub(decimals,rmzeros))
  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("%f %f %f %f", t[1], t[2], t[3], t[4])
    opts.xshift = opts.xshift or format("%f",t[5])
    opts.yshift = opts.yshift or format("%f",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 ""
  optres = optres .. gather_resources(optres)
  local patterns = pdfetcs.patterns
  if pdfmode then
    if opts.bbox then
      attr[#attr+1] = format("/BBox[%s]", opts.bbox)
    end
    attr = tableconcat(attr) :gsub(decimals,rmzeros)
    local index = tex.saveboxresource(boxid, attr, optres, true, opts.bbox and 4 or 1)
    patterns[name] = { id = index, colored = opts.colored }
  else
    local cnt = #patterns + 1
    local objname = "@mplibpattern" .. cnt
    local metric = format("bbox %s", opts.bbox or format("0 0 %s %s",wd,hd))
    texsprint {
      "\\expandafter\\newbox\\csname luamplib.patternbox.", cnt, "\\endcsname",
      "\\global\\setbox\\csname luamplib.patternbox.", cnt, "\\endcsname",
      "\\hbox{\\unhbox ", boxid, "}\\luamplibatnextshipout{",
      "\\special{pdf:bcontent}",
      "\\special{pdf:bxobj ", objname, " ", metric, "}",
      "\\raise\\dp\\csname luamplib.patternbox.", cnt, "\\endcsname",
      "\\box\\csname luamplib.patternbox.", cnt, "\\endcsname",
      "\\special{pdf:put @resources <<", optres, ">>}",
      "\\special{pdf:exobj <<", tableconcat(attr), ">>}",
      "\\special{pdf:econtent}}",
    }
    patterns[cnt] = objname
    patterns[name] = { id = cnt, colored = opts.colored }
  end
end
local function pattern_colorspace (cs)
  local on, new = update_pdfobjs(format("[/Pattern %s]", cs))
  if new then
    local key, val = format("MPlibCS%i",on), format(pdfetcs.resfmt,on)
    if pdfmanagement then
      texsprint {
        "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/ColorSpace}{", key, "}{", val, "}"
      }
    else
      local res = format("/%s %s", key, val)
      if is_defined(pdfetcs.pgfcolorspace) then
        texsprint { "\\csname ", pdfetcs.pgfcolorspace, "\\endcsname{", res, "}" }
      else
        pdfetcs.fallback_update_resources("ColorSpace",res,"@MPlibCS")
      end
    end
  end
  return on
end
local function do_preobj_PAT(object, prescript)
  local name = prescript and prescript.mplibpattern
  if not name then return end
  local patterns = pdfetcs.patterns
  local patt = patterns[name]
  local index = patt and patt.id or err("cannot get pattern object '%s'", name)
  local key = format("MPlibPt%s",index)
  if patt.colored then
    pdf_literalcode("/Pattern cs /%s scn", key)
  else
    local color = prescript.mpliboverridecolor
    if not color then
      local t = object.color
      color = t and #t>0 and luamplib.colorconverter(t)
    end
    if not color then return end
    local cs
    if color:find" cs " or color:find"@pdf.obj" then
      local t = color:explode()
      if pdfmode then
        cs = format("%s 0 R", ltx.pdf.object_id( t[1]:sub(2,-1) ))
        color = t[3]
      else
        cs = t[2]
        color = t[3]:match"%[(.+)%]"
      end
    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]
    add_pattern_resources(key,val)
  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
  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)
  bbox = format("0 0 %f %f", bbox[3]+dx, bbox[4]+dy)
  local streamtext = format("q /Pattern cs/MPlibFd%s scn %s re f Q", on, bbox)
    :gsub(decimals,rmzeros)
  os = format("<</Pattern<</MPlibFd%s %s>>>>", on, format(pdfetcs.resfmt, on))
  on = update_pdfobjs(os)
  local resources = format(pdfetcs.resfmt, on)
  on = update_pdfobjs"<</S/Transparency/CS/DeviceGray>>"
  local attr = tableconcat{
    "/Subtype/Form",
    "/BBox[", bbox, "]",
    "/Matrix[1 0 0 1 ", format("%f %f", -dx,-dy), "]",
    "/Resources ", resources,
    "/Group ", format(pdfetcs.resfmt, on),
  } :gsub(decimals,rmzeros)
  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 = { shifts = { } }
luamplib.trgroupshifts = pdfetcs.tr_group.shifts
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.name = prescript.mplibgroupname or "lastmplibgroup"
    trgroup.isolated, trgroup.knockout = false, false
    for _,v in ipairs(prescript.gr_type:explode",+") do
      trgroup[v] = true
    end
    trgroup.bbox = prescript.mplibgroupbbox:explode":"
    put2output[[\begingroup\setbox\mplibscratchbox\hbox\bgroup]]
  elseif grstate == "stop" then
    local llx,lly,urx,ury = tableunpack(trgroup.bbox)
    put2output(tableconcat{
      "\\egroup",
      format("\\wd\\mplibscratchbox %fbp", urx-llx),
      format("\\ht\\mplibscratchbox %fbp", ury-lly),
      "\\dp\\mplibscratchbox 0pt",
    })
    local grattr = format("/Group<</S/Transparency/I %s/K %s>>",trgroup.isolated,trgroup.knockout)
    local res = gather_resources()
    local bbox = format("%f %f %f %f", llx,lly,urx,ury) :gsub(decimals,rmzeros)
    if pdfmode then
      put2output(tableconcat{
        "\\saveboxresource type 2 attr{/Type/XObject/Subtype/Form/FormType 1",
        "/BBox[", bbox, "]", grattr, "} resources{", res, "}\\mplibscratchbox",
        "\\luamplibtagasgroupbegin",
        [[\setbox\mplibscratchbox\hbox{\useboxresource\lastsavedboxresourceindex}]],
        [[\wd\mplibscratchbox 0pt\ht\mplibscratchbox 0pt\dp\mplibscratchbox 0pt]],
        [[\box\mplibscratchbox]],
        "\\luamplibtagasgroupend",
        "\\endgroup",
        "\\expandafter\\xdef\\csname luamplib.group.", trgroup.name, "\\endcsname{",
        "\\setbox\\mplibscratchbox\\hbox{\\hskip",-llx,"bp\\raise",-lly,"bp\\hbox{",
        "\\useboxresource \\the\\lastsavedboxresourceindex",
        "}}\\wd\\mplibscratchbox",urx-llx,"bp\\ht\\mplibscratchbox",ury-lly,"bp",
        "\\box\\mplibscratchbox}",
      })
    else
      trgroup.cnt = (trgroup.cnt or 0) + 1
      local objname = format("@mplibtrgr%s", trgroup.cnt)
      put2output(tableconcat{
        "\\special{pdf:bxobj ", objname, " bbox ", bbox, "}",
        "\\unhbox\\mplibscratchbox",
        "\\special{pdf:put @resources <<", res, ">>}",
        "\\special{pdf:exobj <<", grattr, ">>}",
        "\\special{pdf:uxobj ", objname, "}",
        "\\endgroup",
      })
      token.set_macro("luamplib.group."..trgroup.name, tableconcat{
        "\\setbox\\mplibscratchbox\\hbox{\\hskip",-llx,"bp\\raise",-lly,"bp\\hbox{",
        "\\special{pdf:uxobj ", objname, "}",
        "}}\\wd\\mplibscratchbox",urx-llx,"bp\\ht\\mplibscratchbox",ury-lly,"bp",
        "\\box\\mplibscratchbox",
      }, "global")
    end
    trgroup.shifts[trgroup.name] = { llx, lly }
  end
  return grstate
end
function luamplib.registergroup (boxid, name, opts)
  local box = texgetbox(boxid)
  local wd, ht, dp = node.getwhd(box)
  local res = (opts.resources or "") .. gather_resources()
  local attr = { "/Type/XObject/Subtype/Form/FormType 1" }
  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")
    opts.matrix = format("%f %f %f %f %f %f",tableunpack(luamplib.transformmatrix))
  end
  local grtype = 3
  if opts.bbox then
    attr[#attr+1] = format("/BBox[%s]", opts.bbox)
    grtype = 2
  end
  if opts.matrix then
    attr[#attr+1] = format("/Matrix[%s]", opts.matrix)
    grtype = opts.bbox and 4 or 1
  end
  if opts.asgroup then
    local t = { isolated = false, knockout = false }
    for _,v in ipairs(opts.asgroup:explode",+") do t[v] = true end
    attr[#attr+1] = format("/Group<</S/Transparency/I %s/K %s>>", t.isolated, t.knockout)
  end
  local trgroup = pdfetcs.tr_group
  trgroup.shifts[name] = { get_macro'MPllx', get_macro'MPlly' }
  local whd
  if pdfmode then
    attr = tableconcat(attr) :gsub(decimals,rmzeros)
    local index = tex.saveboxresource(boxid, attr, res, true, grtype)
    token.set_macro("luamplib.group."..name, tableconcat{
      "\\useboxresource ", index,
    }, "global")
    whd = format("%.3f %.3f 0", wd/factor, (ht+dp)/factor) :gsub(decimals,rmzeros)
  else
    trgroup.cnt = (trgroup.cnt or 0) + 1
    local objname = format("@mplibtrgr%s", trgroup.cnt)
    texsprint {
      "\\expandafter\\newbox\\csname luamplib.groupbox.", trgroup.cnt, "\\endcsname",
      "\\global\\setbox\\csname luamplib.groupbox.", trgroup.cnt, "\\endcsname",
      "\\hbox{\\unhbox ", boxid, "}\\luamplibatnextshipout{",
      "\\special{pdf:bcontent}",
      "\\special{pdf:bxobj ", objname, " width ", wd, "sp height ", ht, "sp depth ", dp, "sp}",
      "\\unhbox\\csname luamplib.groupbox.", trgroup.cnt, "\\endcsname",
      "\\special{pdf:put @resources <<", res, ">>}",
      "\\special{pdf:exobj <<", tableconcat(attr), ">>}",
      "\\special{pdf:econtent}}",
    }
    token.set_macro("luamplib.group."..name, tableconcat{
      "\\setbox\\mplibscratchbox\\hbox{\\special{pdf:uxobj ", objname, "}}",
      "\\wd\\mplibscratchbox ", wd, "sp",
      "\\ht\\mplibscratchbox ", ht, "sp",
      "\\dp\\mplibscratchbox ", dp, "sp",
      "\\box\\mplibscratchbox",
    }, "global")
    whd = format("%.3f %.3f %.3f", wd/factor, ht/factor, dp/factor) :gsub(decimals,rmzeros)
  end
  info("w/h/d of group '%s': %s", name, whd)
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 10 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
            local pattern_ = do_preobj_PAT(object,prescript) -- tiling pattern
            local shading_ = do_preobj_shading(object,prescript) -- shading pattern
            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
              local evenodd, collect, both = false, false, false
              local postscript = object.postscript
              if not object.istext then
                if postscript == "evenodd" then
                  evenodd = true
                elseif postscript == "collect" then
```

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

Removed from ConTeXt general: color stuff.

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

Added : shading

```
local shade_no = do_preobj_SH(object,prescript) -- shading
if shade_no then
  pdf_literalcode"q /Pattern cs"
  objecttype = false
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
  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
if transformed then
  stop_pdf_code()
end
local path = object.htap
```

How can we generate an htap object? Please let us know if you have succeeded.

```
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%s n /MPlibSh%s sh Q",evenodd and "*" or "",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.

```
\ifcsname ProvidesPackage\endcsname
```

We need LaTeX 2024-06-01 as we use ltx.pdf.object_id when pdfmanagement is loaded. But as fp package does not accept an option, we do not append the date option.

```
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{luamplib}
    [2025/03/20 v2.37.2 mplib package for LuaTeX]
\fi
\ifdefined\newluafunction\else
  \input ltluatex
\fi
```

In DVI mode, a new XObject (mppattern, mplibgroup) must be encapsulated in an \hbox. But this should not affect typesetting. So we use Hook mechanism provided by LaTeX kernel. In Plain, atbegshi.sty is loaded.

```
\ifnum\outputmode=0
  \ifdefined\AddToHookNext
    \def\luamplibatnextshipout{\AddToHookNext{shipout/background}}
    \def\luamplibatfirstshipout{\AddToHook{shipout/firstpage}}
    \def\luamplibateveryshipout{\AddToHook{shipout/background}}
  \else
    \input atbegshi.sty
    \def\luamplibatnextshipout#1{\AtBeginShipoutNext{\AtBeginShipoutAddToBox{#1}}}
    \let\luamplibatfirstshipout\AtBeginShipoutFirst
    \def\luamplibateveryshipout#1{\AtBeginShipout{\AtBeginShipoutAddToBox{#1}}}
  \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}}}
```

use Transparency Group

```
\protected\def\usemplibgroup#1#{\usemplibgroupmain}
\def\usemplibgroupmain#1{%
  \mplibstarttousemplibgroup
  \csname luamplib.group.#1\endcsname
  \mplibstoptousemplibgroup
```

```
}
\def\mplibstarttousemplibgroup{\prependtomplibbox\hbox dir TLT\bgroup}
\def\mplibstoptousemplibgroup{\egroup}
\protected\def\mplibgroup#1{%
  \begingroup
  \def\MPllx{0}\def\MPlly{0}%
  \def\mplibgroupname{#1}%
  \mplibgroupgetnexttok
}
\def\mplibgroupgetnexttok{\futurelet\nexttok\mplibgroupbranch}
\def\mplibgroupskipspace{\afterassignment\mplibgroupgetnexttok\let\nexttok= }
\def\mplibgroupbranch{%
  \ifx [\nexttok
    \expandafter\mplibgroupopts
  \else
    \ifx\mplibsptoken\nexttok
      \expandafter\expandafter\expandafter\mplibgroupskipspace
    \else
      \let\mplibgroupoptions\empty
      \expandafter\expandafter\expandafter\mplibgroupmain
    \fi
  \fi
}
\def\mplibgroupopts[#1]{\def\mplibgroupoptions{#1}\mplibgroupmain}
\def\mplibgroupmain{\setbox\mplibscratchbox\hbox\bgroup\ignorespaces}
\protected\def\endmplibgroup{\egroup
  \directlua{ luamplib.registergroup(
    \the\mplibscratchbox, '\mplibgroupname', {\mplibgroupoptions}
  )}%
  \endgroup
}
```

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
    \ifx [\nexttok
      \expandafter\expandafter\expandafter\mplibgobbleoptsmpfig
    \else
      \expandafter\expandafter\expandafter\mplibmainmpfig
    \fi
  \fi
}
\def\mplibgobbleoptsmpfig[#1]{\mplibmainmpfig}
\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]{\xdef\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][]{%
    \xdef\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
```

Code for tagpdf

```
\def\luamplibtagtextbegin#1{}
\let\luamplibtagtextend\relax
\let\luamplibtagasgroupbegin\relax
\let\luamplibtagasgroupend\relax
\ifcsname SuspendTagging\endcsname\else\endinput\fi
\ifcsname ver@tagpdf.sty\endcsname \else
  \ExplSyntaxOn
  \keys_define:nn{luamplib/notag}
    {
      ,alt          .code:n = { }
      ,actualtext   .code:n = { }
      ,artifact     .code:n = { }
      ,text         .code:n = { }
      ,correct-BBox .code:n = { }
      ,tag          .code:n = { }
      ,debug        .code:n = { }
      ,instance     .code:n = { \tl_gset:Nn \currentmpinstancename {#1} }
      ,instancename .meta:n = { instance = {#1} }
      ,unknown      .code:n = { \tl_gset:Ne \currentmpinstancename {\l_keys_key_str} }
    }
  \RenewDocumentCommand\mplibcode{O{}}
    {
      \tl_gset_eq:NN \currentmpinstancename \c_empty_tl
      \keys_set:ne{luamplib/notag}{#1}
      \mplibtmptoks{}\ltxdomplibcode
    }
  \ExplSyntaxOff
  \let\mplibalttext \luamplibtagtextbegin
  \let\mplibactualtext \mplibalttext
  \endinput\fi
\let\mplibstarttoPDForiginal\mplibstarttoPDF
\let\mplibstoptoPDForiginal\mplibstoptoPDF
\let\mplibputtextboxoriginal\mplibputtextbox
\let\mplibstarttousemplibgrouporiginal\mplibstarttousemplibgroup
\let\mplibstoptousemplibgrouporiginal\mplibstoptousemplibgroup
\ExplSyntaxOn
\tl_new:N \l__luamplib_tag_alt_tl
\tl_new:N \l__luamplib_tag_alt_dflt_tl
\tl_set:Nn\l__luamplib_tag_alt_dflt_tl {metapost~figure}
\tl_new:N \l__luamplib_tag_actual_tl
\tl_new:N \l__luamplib_tag_struct_tl
\tl_set:Nn\l__luamplib_tag_struct_tl {Figure}
\bool_new:N \l__luamplib_tag_usetext_bool
\bool_new:N \l__luamplib_tag_BBox_bool
\bool_set_true:N \l__luamplib_tag_BBox_bool
\seq_new:N\l__luamplib_tag_bboxcorr_seq
```

```
\bool_new:N\l__luamplib_tag_bboxcorr_bool
\bool_new:N \l__luamplib_tag_debug_bool
\tl_new:N \l__luamplib_BBox_label_tl
\tl_new:N \l__luamplib_BBox_llx_tl
\tl_new:N \l__luamplib_BBox_lly_tl
\tl_new:N \l__luamplib_BBox_urx_tl
\tl_new:N \l__luamplib_BBox_ury_tl
\cs_set_nopar:Npn \luamplibtagtextbegin #1
{
  \bool_if:NTF \l__luamplib_tag_usetext_bool
  {
    \tag_mc_end_push:
    \tag_mc_begin:n{}
    \tag_struct_begin:n{tag=NonStruct,stash}
    \def\myboxnum{#1}
    \edef\mystructnum{\tag_get:n{struct_num}}
    \edef\statebeforebox{\inteval{\tag_get:n{struct_counter}+\tag_get:n{mc_counter}}}
  }
  {
    \tag_if_active:TF
      { \bool_set_true:N \l_tmpa_bool }
      { \bool_set_false:N \l_tmpa_bool }
    \SuspendTagging{luamplib.textext}
  }
}
\cs_set_nopar:Npn \luamplibtagtextend
{
  \bool_if:NTF \l__luamplib_tag_usetext_bool
  {
    \edef\stateafterbox{\inteval{\tag_get:n{struct_counter}+\tag_get:n{mc_counter}}}
    \tag_if_active:T {
      \int_compare:nNnTF
      {\stateafterbox}
      =
      {\statebeforebox}
      { \cs_gset_nopar:cpe {luamplib.notagbox.\myboxnum} {\mystructnum} }
      { \cs_gset_nopar:cpe {luamplib.tagbox.\myboxnum} {\mystructnum} }
    }
    \tag_struct_end:
    \tag_mc_end:
    \tag_mc_begin_pop:n{}
  }
  {
    \bool_if:NT \l_tmpa_bool
      { \ResumeTagging{luamplib.textext} }
  }
}
\msg_new:nnn {luamplib}{figure-text-reuse}
{
  textext~box~#1~probably~is~incorrectly~tagged.\\
  Reusing~a~box~in~text-keyed~figures~is~strongly~discouraged.
}
\cs_set_nopar:Npn \mplibputtextbox #1
{
  \vbox to 0pt{\vss\hbox to 0pt{%
    \bool_if:NTF \l__luamplib_tag_usetext_bool
    {
      \ResumeTagging{luamplib.puttextbox}
      \tag_mc_end:
      \cs_if_exist:cTF {luamplib.tagbox.#1}
      {
        \tag_struct_use_num:n {\csname luamplib.tagbox.#1\endcsname}
        \raise\dp#1\copy#1
      }
      {
        \cs_if_exist:cF {luamplib.notagbox.#1}
        {
          \msg_warning:nnn{luamplib}{figure-text-reuse}{#1}
        }
        \tag_mc_begin:n{}
        \int_set:Nn \l_tmpa_int {#1}
        \tag_mc_reset_box:N \l_tmpa_int
        \raise\dp#1\copy#1
        \tag_mc_end:
      }
      \tag_mc_begin:n{artifact}
    }
    {
      \int_set:Nn \l_tmpa_int {#1}
      \tag_mc_reset_box:N \l_tmpa_int
      \raise\dp#1\copy#1
    }
  \hss}}
}
\cs_new_nopar:Npn \__luamplib_tagging_begin_figure:
{
  \tag_if_active:T
  {
    \tag_mc_end_push:
    \tl_if_empty:NT\l__luamplib_tag_alt_tl
    {
      \msg_warning:nne{luamplib}{alt-text-missing}{\l__luamplib_tag_alt_dflt_tl}
      \tl_set:Ne\l__luamplib_tag_alt_tl {\l__luamplib_tag_alt_dflt_tl}
    }
    \tag_struct_begin:n
    {
      tag=\l__luamplib_tag_struct_tl,
      alt=\l__luamplib_tag_alt_tl,
    }
    \tag_mc_begin:n{}
  }
}
\cs_new_nopar:Npn \__luamplib_tagging_end_figure:
{
  \tag_if_active:T
  {
    \tag_mc_end:
    \tag_struct_end:
```

```
    \tag_mc_begin_pop:n{}
  }
}
\cs_new_nopar:Npn \__luamplib_tagging_begin_actualtext:
{
  \tag_if_active:T
  {
    \tag_mc_end_push:
    \tag_struct_begin:n
    {
      tag=Span,
      actualtext=\l__luamplib_tag_actual_tl,
    }
    \tag_mc_begin:n{}
  }
}
\cs_set_eq:NN \__luamplib_tagging_end_actualtext: \__luamplib_tagging_end_figure:
\cs_new_nopar:Npn \__luamplib_tagging_begin_artifact:
{
  \tag_if_active:T
  {
    \tag_mc_end_push:
    \tag_mc_begin:n{artifact}
  }
}
\cs_new_nopar:Npn \__luamplib_tagging_end_artifact:
{
  \tag_if_active:T
  {
    \tag_mc_end:
    \tag_mc_begin_pop:n{}
  }
}
\cs_set_eq:NN \luamplibtaggingbegin \__luamplib_tagging_begin_figure:
\cs_set_eq:NN \luamplibtaggingend \__luamplib_tagging_end_figure:
\keys_define:nn{luamplib/tag}
  {
    ,alt .code:n =
      {
        \tl_set:Ne\l__luamplib_tag_alt_tl{\text_purify:n{#1}}
      }
    ,actualtext .code:n =
      {
        \bool_set_false:N \l__luamplib_tag_BBox_bool
        \tl_set:Ne\l__luamplib_tag_actual_tl{\text_purify:n{#1}}
        \cs_set_eq:NN \luamplibtaggingbegin \__luamplib_tagging_begin_actualtext:
        \cs_set_eq:NN \luamplibtaggingend \__luamplib_tagging_end_actualtext:
        \tag_if_active:T {\noindent}
      }
    ,artifact .code:n =
      {
        \bool_set_false:N \l__luamplib_tag_BBox_bool
        \cs_set_eq:NN \luamplibtaggingbegin \__luamplib_tagging_begin_artifact:
        \cs_set_eq:NN \luamplibtaggingend \__luamplib_tagging_end_artifact:
      }
    ,text .code:n =
      {
        \bool_set_false:N \l__luamplib_tag_BBox_bool
        \bool_set_true:N  \l__luamplib_tag_usetext_bool
        \cs_set_eq:NN \luamplibtaggingbegin \__luamplib_tagging_begin_artifact:
        \cs_set_eq:NN \luamplibtaggingend \__luamplib_tagging_end_artifact:
        \tag_if_active:T {\noindent}
      }
    ,tag .code:n =
      {
        \str_case:nnF {#1}
          {
            {text}
            {
              \bool_set_false:N \l__luamplib_tag_BBox_bool
              \bool_set_true:N  \l__luamplib_tag_usetext_bool
              \cs_set_eq:NN \luamplibtaggingbegin \__luamplib_tagging_begin_artifact:
              \cs_set_eq:NN \luamplibtaggingend \__luamplib_tagging_end_artifact:
              \tag_if_active:T {\noindent}
            }
            {false}
            {
              \SuspendTagging{luamplib.tagfalse}
            }
          }
          {
            \tl_set:Nn\l__luamplib_tag_struct_tl{#1}
          }
      }
    ,correct-BBox .code:n =
      {
        \bool_set_true:N \l__luamplib_tag_bboxcorr_bool
        \seq_set_split:Nnn \l__luamplib_tag_bboxcorr_seq{~}{#1~0pt~0pt~0pt~0pt}
      }
    ,debug .code:n =
      { \bool_set_true:N \l__luamplib_tag_debug_bool }
    ,instance .code:n =
      { \tl_gset:Nn \currentmpinstancename {#1} }
    ,instancename .meta:n = { instance = {#1} }
    ,unknown .code:n =
      { \tl_gset:Ne \currentmpinstancename {\l_keys_key_str} }
   }
\cs_new_nopar:Npn \luamplibtaggingBBox
{
  \bool_lazy_and:nnT
  {\tag_if_active_p:}
  {\l__luamplib_tag_BBox_bool}
  {
    \tl_set:Ne \l__luamplib_BBox_label_tl {luamplib.BBox.\tag_get:n{struct_num}}
    \tex_savepos:D
    \property_record:ee{\l__luamplib_BBox_label_tl}{xpos,ypos,abspage}
    \tl_set:Ne \l__luamplib_BBox_llx_tl
        {
```

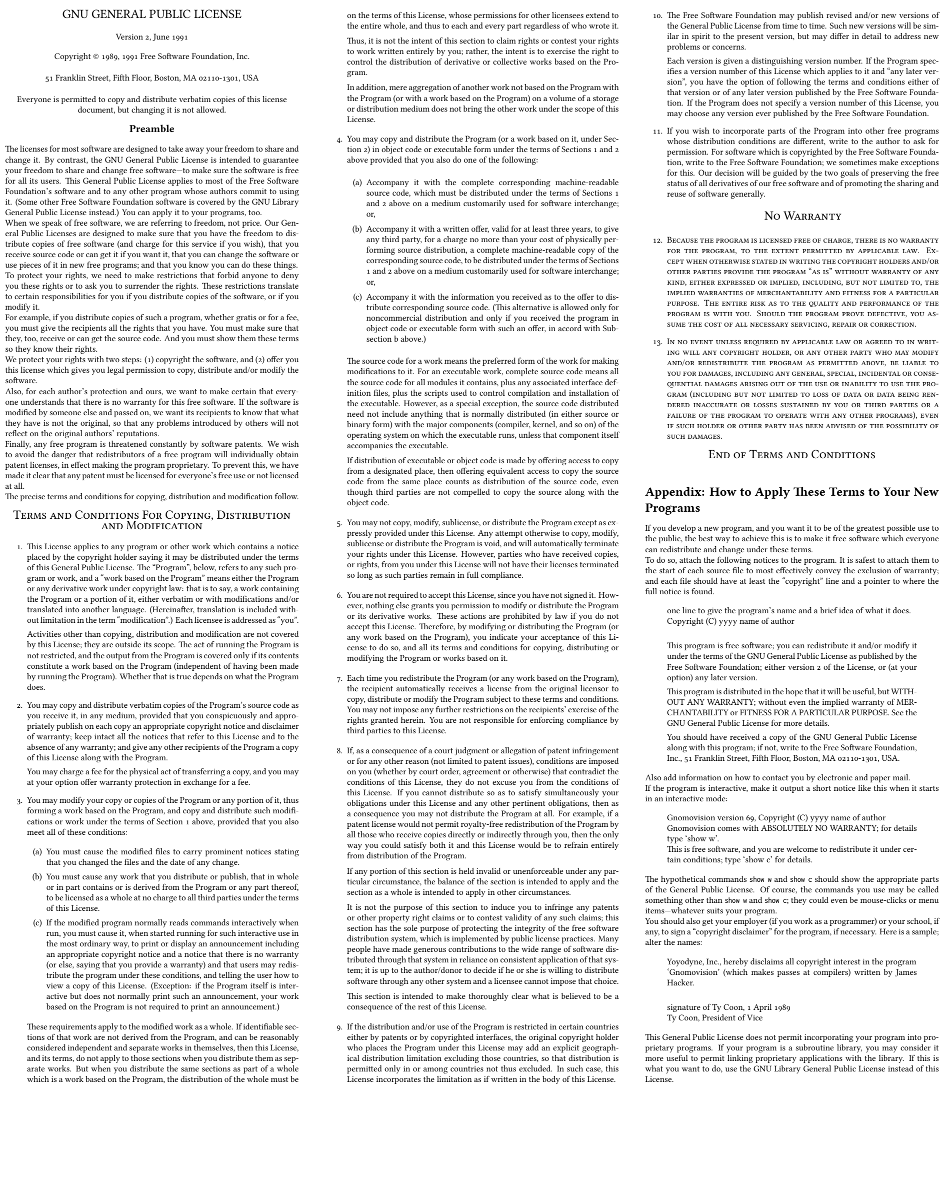
```
      \dim_to_decimal_in_bp:n
      { \property_ref:een {\l__luamplib_BBox_label_tl}{xpos}{0}sp }
    }
  \tl_set:Ne \l__luamplib_BBox_lly_tl
    {
      \dim_to_decimal_in_bp:n
      { \property_ref:een {\l__luamplib_BBox_label_tl}{ypos}{0}sp - \dp\mplibscratchbox }
    }
  \tl_set:Ne \l__luamplib_BBox_urx_tl
    {
      \dim_to_decimal_in_bp:n
      { \l__luamplib_BBox_llx_tl bp + \wd\mplibscratchbox }
    }
  \tl_set:Ne \l__luamplib_BBox_ury_tl
    {
      \dim_to_decimal_in_bp:n
      { \l__luamplib_BBox_lly_tl bp + \ht\mplibscratchbox + \dp\mplibscratchbox }
    }
  \bool_if:NT \l__luamplib_tag_bboxcorr_bool
    {
      \tl_set:Ne \l__luamplib_BBox_llx_tl
        {
          \fp_eval:n
          {
            \l__luamplib_BBox_llx_tl
            +
            \dim_to_decimal_in_bp:n {\seq_item:Nn \l__luamplib_tag_bboxcorr_seq {1} }
          }
        }
      \tl_set:Ne \l__luamplib_BBox_lly_tl
        {
          \fp_eval:n
          {
            \l__luamplib_BBox_lly_tl
            +
            \dim_to_decimal_in_bp:n {\seq_item:Nn \l__luamplib_tag_bboxcorr_seq {2} }
          }
        }
      \tl_set:Ne \l__luamplib_BBox_urx_tl
        {
          \fp_eval:n
          {
            \l__luamplib_BBox_urx_tl
            +
            \dim_to_decimal_in_bp:n {\seq_item:Nn \l__luamplib_tag_bboxcorr_seq {3} }
          }
        }
      \tl_set:Ne \l__luamplib_BBox_ury_tl
        {
          \fp_eval:n
          {
            \l__luamplib_BBox_ury_tl
            +
            \dim_to_decimal_in_bp:n {\seq_item:Nn \l__luamplib_tag_bboxcorr_seq {4} }
          }
        }
      }
    \prop_gput:cne
      { g__tag_struct_\tag_get:n{struct_num}_prop }
      {A}
      {
        << /O /Layout /BBox [
          \l__luamplib_BBox_llx_tl\c_space_tl
          \l__luamplib_BBox_lly_tl\c_space_tl
          \l__luamplib_BBox_urx_tl\c_space_tl
          \l__luamplib_BBox_ury_tl
        ] >>
      }
    \bool_if:NT \l__luamplib_tag_debug_bool
      {
        \iow_log:e
          {
            luamplib/tag/debug:~BBox~of~structure~\tag_get:n{struct_num}~is~
            \l__luamplib_BBox_llx_tl\c_space_tl
            \l__luamplib_BBox_lly_tl\c_space_tl
            \l__luamplib_BBox_urx_tl\c_space_tl
            \l__luamplib_BBox_ury_tl
          }
        \use:e
        {
          \exp_not:N\AddToHookNext{shipout/foreground}
          {
            \exp_not:N\int_compare:nNnT
            {\exp_not:N\g_shipout_readonly_int}
            =
            {\property_ref:een{\l__luamplib_BBox_label_tl}{abspage}{0}}
            {
              \exp_not:N\put
              (\l__luamplib_BBox_llx_tl bp, \dim_eval:n{\l__luamplib_BBox_lly_tl bp -\paperheight})
              {
                \exp_not:N\opacity_select:n{0.5} \exp_not:N\color_select:n{red}
                \exp_not:N\rule
                  {\dim_eval:n {\l__luamplib_BBox_urx_tl bp - \l__luamplib_BBox_llx_tl bp}}
                  {\dim_eval:n {\l__luamplib_BBox_ury_tl bp - \l__luamplib_BBox_lly_tl bp}}
              }
            }
          }
        }
      }
  }
}
\cs_set_nopar:Npn \luamplibtagasgroupbegin
{
  \bool_if:NT \l__luamplib_tag_usetext_bool
  {
    \ResumeTagging{luamplib.asgroup}
    \tag_mc_begin:n{}
  }
```

```
}
\cs_set_nopar:Npn \luamplibtagasgroupend
{
  \bool_if:NT \l__luamplib_tag_usetext_bool
  {
    \tag_mc_end:
    \SuspendTagging{luamplib.asgroup}
  }
}
\cs_set_nopar:Npn \mplibstarttousemplibgroup
{
  \prependtomplibbox\hbox dir TLT\bgroup
  \luamplibtaggingbegin
  \setbox\mplibscratchbox\hbox\bgroup
  \bool_if:NT \l__luamplib_tag_usetext_bool
  {
    \tag_mc_end:
    \tag_mc_begin:n{}
  }
}
\cs_set_nopar:Npn \mplibstoptousemplibgroup
{
  \bool_if:NT \l__luamplib_tag_usetext_bool
  {
    \tag_mc_end:
    \tag_mc_begin:n{artifact}
  }
  \egroup
  \luamplibtaggingBBox
  \unhbox\mplibscratchbox
  \luamplibtaggingend
  \egroup
}
\cs_set_nopar:Npn \mplibstarttoPDF #1 #2 #3 #4
  {
    \prependtomplibbox
    \hbox dir TLT\bgroup
    \luamplibtaggingbegin % begin tagging
    \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
    \SuspendTagging{luamplib.mplibtopdf}% stop tag inside figure
    \noindent
  }
\cs_set_nopar:Npn \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
    \luamplibtaggingBBox % BBox
    \box\mplibscratchbox
    \luamplibtaggingend % end tagging
    \egroup
  }
\RenewDocumentCommand\mplibcode{O{}}
  {
    \msg_set:nnn {luamplib}{alt-text-missing}
    {
      Alternative~text~for~mplibcode~is~missing.\\
      Using~the~default~value~'##1'~instead.
    }
    \tl_gset_eq:NN \currentmpinstancename \c_empty_tl
    \keys_set:ne{luamplib/tag}{#1}
    \tl_if_empty:NF \currentmpinstancename
      { \tl_set:Nn\l__luamplib_tag_alt_dflt_tl {metapost~figure~\currentmpinstancename} }
    \mplibtmptoks{}\ltxdomplibcode
  }
\RenewDocumentCommand\mpfig{s O{}}
  {
    \begingroup
    \IfBooleanTF{#1}
    {\mplibprempfig *}
    {
      \msg_set:nnn {luamplib}{alt-text-missing}
      {
        Alternative~text~for~mpfig~is~missing.\\
        Using~the~default~value~'##1'~instead.
      }
      \keys_set:ne{luamplib/tag}{#2}
      \tl_if_empty:NF \mpfiginstancename
        { \tl_set:Nn\l__luamplib_tag_alt_dflt_tl {metapost~figure~\mpfiginstancename} }
      \mplibmainmpfig
    }
  }
\RenewDocumentCommand\usemplibgroup{O{} m}
  {
    \begingroup
    \msg_set:nnn {luamplib}{alt-text-missing}
    {
      Alternative~text~for~usemplibgroup~is~missing.\\
      Using~the~default~value~'##1'~instead.
    }
    \keys_set:ne{luamplib/tag}{#1}
    \tl_set:Nn\l__luamplib_tag_alt_dflt_tl {metapost~figure~#2}
    \mplibstarttousemplibgroup
    \csname luamplib.group.#2\endcsname
    \mplibstoptousemplibgroup
    \endgroup
  }
\cs_new_nopar:Npn \mplibalttext #1
{
  \tl_set:Ne \l__luamplib_tag_alt_tl {\text_purify:n{#1}}
}
\cs_new_nopar:Npn \mplibactualtext #1
{
  \tl_set:Ne \l__luamplib_tag_actual_tl {\text_purify:n{#1}}
}
\ExplSyntaxOff
```

That's all folks!

3 The GNU GPL License v2

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



GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

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

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

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.
The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

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 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

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

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

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> 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.