The luamplib package

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

2026/05/05 v2.41.1

Abstract

Package to have METAPOST code typeset directly in a document with LuaTEX

Contents

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 to run a TEX code. VerbatimTeX ⟨*string*⟩ is a more versatile macro corresponding to verbatimtex command. Of course the behavior cannot be the same as the stand-alone **mpost**, so that 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 § 1.1.6.

- In the past, the package required PDF mode in order to have some output. Starting with v2.7 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 that \centering, \raggedleft etc. will have effects. \mplibnoforcehmode, being default for backward compatibility, reverts this setting.[1]

1.1.2 \everymplib{...}, \everyendmplib{...}

\everymplib and \everyendmplib redefine the Lua table entry 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 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 § 1.2.11). You can try other effects as well, though we did not fully tested their proper functioning.

[1] Actually these commands redefine \prependtomplibbox. So you can redefine this macro with anything suitable before a box. But see § 1.1.18 on Tagged PDF.

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=⟨numeric⟩"` to the sentence. ($0 \leq \langle numeric \rangle \leq 1$)

From v2.36, `withtransparency` is available with *plain* format as well. See below § 1.2.4.

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 § 1.2.6.

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 and Foxit Editor, cannot properly render the isolated or knockout effect.

Transparency group is available with *plain* format as well with extended functionality. See below § 1.2.13.

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}`[2] 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}`

Legacy behavior 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.[3]

```
\mplibcode
  verbatimtex \moveright 3cm etex; beginfig(0); ... endfig;
```

[2] As for user's setting, `enable`, `true` and `yes` are identical; all others are identical to `disable`.

[3] But the recommended way to reuse a figure is using `\mplibgroup` command. See below § 1.2.14.

```
  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 mplibcode environment.

On the other hand, TEX code in `verbatimtex ... etex` between `beginfig()` and `endfig` will be inserted after flushing out the METAPOST figure. An example:[4]

```
\mplibcode
  D := sqrt(2)**9;
  beginfig(0);
    draw fullcircle scaled D;
    VerbatimTeX("\gdef\Dia{" & decimal D & "}");
  endfig;
\endmplibcode
diameter: \Dia bp.
```

diameter: 22.62764bp.

New and recommended way By contrast, when `\mpliblegacybehavior{disable}` is declared, any `verbatimtex ... etex`, along with `btex ... etex`, will be run sequentially one by one. So, some TEX code in `verbatimtex ... etex` will have effect on `btex ... etex` codes thereafter.

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

ABC **DEF** **GHI**

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.

[4] But the recommended way to access METAPOST variables from TEX (or Lua) side is to use Lua code via `luamplib.instances`. For details see below § 1.3.2.

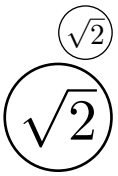
1.1.8 \mplibcodeinherit{enable|disable}

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

1.1.9 \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. The 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
```



1.1.10 Separate METAPOST instances

luamplib v2.22 has added the support for several named METAPOST instances in LaTeX environment mplibcode or Plain TeX commands \mplibcode ... \endmplibcode. 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 the 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]{...}
```

1.1.11 \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 § 1.1.12 and § 1.1.13), 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.

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


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 command, in principle). See the example above at § 1.1.12. The optional [...] denotes the option of **xcolor**'s \color command. For spot colors, **l3color** module is well supported in PDF and DVI mode. Package **colorspace** is supported as well in PDF mode, but could conflict with **luamplib**'s special features such as shading when \DocumentMetadata, ie. PDF management code, is not loaded.

N.B. Formerly, only the first object would have been colored as intended among multiple graphical objects in a METAPOST image, because \mpcolor always produced withprescript command internally. Since v2.38.1, now that \mpcolor returns a METAPOST color expression if possible, users can issue the sentence as follows without worrying about the location of the color command:

```
draw image (drawarrow (left--right) scaled 5)
  scaled 8
  withcolor \mpcolor{red!50} ;
```

N.B. Be aware, however, that even after v2.38.1 \mpcolor still inserts withprescript command when the color specified is a spot color (or named color in DVI mode). Users therefore have to revise the code so that the color can have effect inside the image. For instance:

```
draw image (drawarrow (left--right) scaled 5)
  scaled 8
  withcolor \mpcolor{spotA}
  withoutcolor ;
```

or preferably,

```
draw image (drawarrow (left--right) scaled 5 withcolor \mpcolor{spotA})
  scaled 8 ;
```

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 1/3[red,white]); }
\mpfig* input boxes \endmpfig
\mpfig
  circleit.a(btex Box 1 etex); drawboxed(a);
\endmpfig
```

Users can change the instance name (default value: @mpfig) 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 is not true.

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 neceeccsary before returning their paths to the 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 without .mp extension. Note that .mp files under $TEXMFMAIN/metapost/base and $TEXMFMAIN/metapost/context/base are already registered by default.

N.B. \mplibmakenocache{*} will suppress making cache files. Use it at your own risk.

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. Available optional keys are similar to those of the LaTeX's picture environment (texdoc latex-lab-graphic). The default tagging mode is the alt key with Figure structure.

alt=⟨*text*⟩ starts a Figure tag by default and sets an alternate text of the figure from the ⟨*text*⟩. BBox info will be added automatically to the PDF. This key is needed for ordinary METAPOST figures, for which, if no alt text is given, a default text will be used with a warning issued. You can change the alternate text within METAPOST code as well: VerbatimTeX "\mplibalttext{⟨*text*⟩}";

actualtext=⟨*text*⟩ starts a Span tag implicitly and sets a replacement text (a.k.a. actual text) from the ⟨*text*⟩. If in vertical mode, horizontal mode will be forced by \noindent command.[5] 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 change the actual text within METAPOST code as well: VerbatimTeX "\mplibactualtext{⟨*text*⟩}";

[5]It is not recommended to personally redefine \prependtomplibbox. Apart from using \mplibforcehmode or \mplibnoforcehmode, the redefinition might be incompatible with actualtext key. See § 1.1.1 on these commands.

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

text starts an `Artifact` MC but enables tagging on TEX-text boxes (such as `btex ... etex`, excluding pictures made by `infont` operator). If in vertical mode, horizontal mode will be forced by `\noindent` command.[6] BBox info will not be added. This key is intended for figures the meaning of which is the sequence of texts in the TEX-text boxes in the order they are drawn in the figure.

N.B. Within `text`-mode figures, reusing TEX-text boxes is strongly discouraged.

Note that the text in a TEX-text box which starts with `[taggingoff]` will not be tagged at all, and of course `[taggingoff]` and its trailing spaces will be gobbled by **luamplib**. For example, the first and the third boxes in the following figure will not be tagged, and still remain in the `Artifact` MC-chunks.

```
\begin{mplibcode}[text]
  beginfig(1)
    draw btex [taggingoff] $\sqrt 2$ etex ;
    draw textext "$\sqrt 3$" shifted 12down ;
    draw TEX "[taggingoff] $\sqrt 5$" shifted 24down ;
    draw maketext "$\sqrt 7$" shifted 36down ;
    draw mplibgraphictext "$\sqrt x$" shifted 48down ;
  endfig;
\end{mplibcode}
```

$\sqrt{2}$
$\sqrt{3}$
$\sqrt{5}$
$\sqrt{7}$
$\sqrt{x}$

off Given this key, nothing will be tagged by **luamplib**.

tag=⟨*name*⟩ You can choose a tag name, default value being `Figure`.[7] For instance, you can set `tag=Formula, alt=`⟨*text*⟩ to get a `Formula` element with its alternate text.[8]

adjust-BBox=⟨*dimens*⟩ You can correct the BBox attribute of the figure by space-separated four dimensional values, which will be added to the automatically calculated BBox values. To draw the bounding box for checking with half-transparent red color, you can add `debug=BBox` to the argument of `\DocumentMetadata` command.

tagging-setup=⟨*key-val list*⟩ This key accepts as its value the list of key-value options mentioned so far.

You can set these options anywhere in the document by declaring `\SetKeys[luamplib/tagging]` {⟨*key-val list*⟩}, which will affect **mplib** figures thereafter in the scope. And the options listed above are provided for `\mpfig` and `\usemplibgroup` (see below § 1.2.13) commands as well.

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

[6] The key `text` also shares the limitation mentioned in the previous footnote.
[7] The option `tag=false`, however, is a synonym of the `off` key.
[8] Beware that this bypasses LATEX's regular math formula tagging, for which the `text` key is needed.

```
\end{mplibcode}

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

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

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

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

1.2 MetaPost

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

`mplibdimen` ⟨*string*⟩ and `mplibcolor` ⟨*string*⟩ are METAPOST interfaces for the TEX commands `\mpdim` and `\mpcolor` (see above § 1.1.12 and § 1.1.13). 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` ⟨*string*⟩ 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` command.[9] 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. 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.

N.B. Spot colors are forced to cmyk or rgb model, so these operators are not recommended for spot colors.

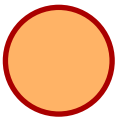
1.2.3 `withmplibcolors (..., ...)`

Unlike the `withcolor` command, users can specify one color for filling and another color for stroking using the macro `withmplibcolors` at the end of a sentence. The syntax is `withmplibcolors`

[9] Since v2.38.1, the operation of `mplibtexcolor` is the same as that of `mplibcolor` if the color specified is not a spot color or a named color in DVI mode.

(⟨*fill color expr*⟩, ⟨*stroke color expr*⟩). When the argument is in string type, it is regarded as the color expression of TEX side. A simple example (see also the example at § 1.2.10):

```
filldraw fullcircle scaled 40
  withpen pencircle scaled 2
  withmplibcolors ("orange!60", 2/3red) ;
```

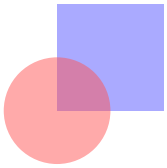


The PDF file size is much smaller than issueing two sentences with different colors, though the apparent effect is the same.

1.2.4 withtransparency (..., ...)

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

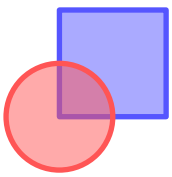
```
\mpfig
  fill unitsquare scaled 40
    withcolor 1/3[blue,white]
    withtransparency (1, 0.5)          % or ("normal", 0.5)
    ;
  fill fullcircle scaled 40
    withcolor 1/3[red,white]
    withtransparency (1, 0.5)
    ;
\endmpfig
```



1.2.5 withmplibopacities (..., ..., ...)

By analogy with the macro withmplibcolors (see above § 1.2.3), the macro withmplibopacities is also provided. The syntax is withmplibopacities (⟨*number*⟩ | ⟨*string*⟩, ⟨*numeric*⟩, ⟨*numeric*⟩). The first argument is the same as that of withtransparency command described above at § 1.2.4; the latter two arguments are numeric expressions denoting *fill opacity* and *stroke opacity* respectively. It is more efficient than issueing two sentences with different opacities.

```
\mpfig
  pickup pencircle scaled 2;
  filldraw unitsquare scaled 40
    withcolor 1/3[blue,white]
    withmplibopacities (1, 1/2, 1)      % or ("normal", 1/2, 1)
    ;
  filldraw fullcircle scaled 40
    withcolor 1/3[red,white]
    withmplibopacities (1, 1/2, 1)
    ;
\endmpfig
```



1.2.6 ... 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* as well as paths can have shading effect. The term *textual picture* here means a picture generated by btex ... etex, textext, TEX, maketext, mplibgraphictext (see below § 1.2.8), or infont operator, though technically only the last one is a true textual picture. Note that the picture, including transparency group, in which the objects are filled *without* color can also be regarded as a textual picture (e.g., see below § 1.2.10, particularly the first example of tiling pattern at § 1.2.12; see also § 1.2.13 and § 1.2.14).

```
draw btex \bfseries\TeX etex rotated 15 scaled 6
  withshadingmethod "linear"
  withshadingvector (0,3)
  withshadingstep (
    withshadingfraction 1/2
    withshadingcolors (red,green)
  )
  withshadingstep (
    withshadingfraction 1
    withshadingcolors (green,blue)
  ) ;
```

- When shading a picture generated by 'infont' operator or that has multiple components, the effect of withshadingvector and that of withshadingdirection will be the same, as luamplib considers only the bounding box of the picture.
- In addition to those mimicking *metafun*'s, a few more optional macros are available: withshadingpoints, withshadingcenters, withshadingextend, and withshadingstroke.

The syntax is ⟨*path*⟩ | ⟨*textual picture*⟩ withshadingmethod ⟨*string*⟩, where the latter shall be either "linear" or "circular". The balance of this subsection is to explain additional optional macros. Above all, there are two ways in specifying the shading coordinates, of which you can choose the more convenient one. First, the way that mimicks the *metafun*'s:

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 being (0,2).

withshadingorigin ⟨***pair***⟩ The center of both starting and ending circles, default value being center p, where p is the operand of withshadingmethod.

withshadingcenter ⟨***pair***⟩ Value to specify the starting center. For instance, (0,0) means that the center of starting circle is center p; (1,1) means urcorner p; (-1,-1) means llcorner 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 ⟨***numeric***⟩ Multiplier of the radii, default value being `1.2`. This is no-op in linear mode.

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

Secondly, the way provided by luamplib only:

withshadingpoints (⟨***pair***⟩**,** ⟨***pair***⟩**)** In linear mode, values to specify directly the starting and ending points: you can use it instead of `withshadingvector` or `withshadingdirection`. In circular mode, the centers of starting and ending circles: it could be easier than issueing two macros `withshadingorigin` and `withshadingcenter`. Note that, within the macro, both `withshadingfactor 1` and `withshadingtransform "no"` are already decalred.

withshadingcenters (⟨***pair***⟩**,** ⟨***pair***⟩**)** Synonym of `withshadingpoints`. Normally accompanied by `withshadingradius` which has the same meaning as described above.

Now, optional macros common to the both ways:

withshadingstep (...**)** For combined shading of more than two colors.

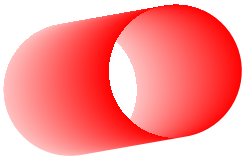
withshadingfraction ⟨***numeric***⟩ Fractional number of each shading step, and so only meaningful within `withshadingstep`.

withshadingcolors (⟨***color expr***⟩**,** ⟨***color expr***⟩**)** Starting and ending colors, default value being `(white, black)`. String-type argument is regarded as the color expression of TEX side.

withshadingdomain ⟨***pair***⟩ Limiting values of parametric variable that varies on the axis of color gradient, default value being `(0,1)`. Of course the values can be negative or greater than 1.

withshadingextend (⟨***boolean***⟩**,** ⟨***boolean***⟩**)** Values specifying whether to extend the shading beyond the starting and ending points or circles, default value being `(true, true)`. An example just to show the concept:

```
\mpfig
  path p[];
  p1 = fullcircle scaled 50;
  p2 = fullcircle scaled 50 shifted 40 right;
  fill (subpath (2,6) of p1 -- subpath (-2,2) of p2 -- cycle) rotated 10
    withshadingmethod  "circular"
    withshadingcenters (center p1, center p2 rotated 10)
    withshadingradius  (25, 25)
    withshadingcolors  (3/4[red,white], red)
    withshadingextend  (false, false) ;
\endmpfig
```



withshadingstroke ⟨***string***⟩ where ⟨*string*⟩ shall be "yes" or "no". Only meaningful when the shading object is a ⟨*path*⟩; if "yes", we get the path stroked and *then* shaded. It is more efficient than issueing two sentences.

1.2.7 ... withfademethod ...

This is a METAPOST command which makes the color of an object gradiently transparent, a.k.a. *fading*. 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 object of fading can be a picture as well as a path; (2) you cannot make gradient colors, but can only make gradient opacity. Technically speaking, this command generates and applies a special kind of masking transparency group described below at § 1.2.15.

Related macros to control optional values are:

withfadeopacity (⟨*numeric*⟩, ⟨*numeric*⟩) 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 (⟨*numeric*⟩, ⟨*numeric*⟩) 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 at § 1.2.13 on the analogous macro withgroupbbox.

An example:

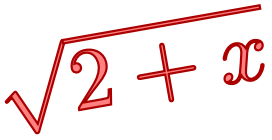
```
draw
  btex \includegraphics[width=100bp]{mill} etex
  withfademethod  "circular"
  withfaderadius  (20, 50)
  withfadeopacity (1, 0) ;
```

1.2.8 mplibgraphictext ...

mplibgraphictext ⟨*string*⟩ is a METAPOST operator, the effect of which is similar to that of ConTEXt's graphictext or our own mpliboutlinetext (see below § 1.2.11). However the syntax is somewhat different.

```
draw mplibgraphictext "$\sqrt{2+x}$"
```

```
rotated 10 scaled 3
fakebold 2.5                        % fontspec option
fillcolor "red!50"                  % color expression
drawcolor 2/3 red                   % or strokecolor 2/3 red
;
```



fakebold, fillcolor and drawcolor (or strokecolor) are optional; default values are 2, "white" and "black" respectively.[10] When the color expression is given in string type, it is regarded as **color**, **xcolor** or **l3color**'s expression. All from mplibgraphictext to the end of sentence will compose an anonymous picture, which can be drawn or assigned to a variable. Incidentally, withfillcolor and withdrawcolor are synonyms of fillcolor and drawcolor, hopefully to be compatible with graphictext.

N.B. In some cases, especially when processing complicated TEX code, mplibgraphictext will produce better results than ConTEXt or even than our own mpliboutlinetext, not to mention the much smaller PDF file size. There are, however, some limitations such that you can't apply shading (gradient colors) to the text with *metafun*'s withshademethod.[11] Again, in DVI mode, **unicode-math** package is needed for math formulae, as we cannot embolden type1 fonts in DVI mode. But the most critical limitation is that, unlike mpliboutlinetext, you cannot manipulate the shape of outline paths, because the returned picture is basically a btex ... etex picture.

1.2.9 mplibglyph ... of ...

METAPOST operator mplibglyph ⟨*number*⟩ | ⟨*string*⟩ of ⟨*number*⟩ | ⟨*string*⟩ returns a METAPOST picture containing outline paths of a glyph in OpenType, TrueType or Type1 (.pfb) fonts. When a TFM 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 "R" of "utmr8a.pfb"                       % raw filename (type1 font)
mplibglyph "Q" of "Times.ttc(2)"                     % subfont number
mplibglyph "Q" of "SourceHanSansK-VF.otf[Regular]"   % instance name
mplibglyph "R" of "SourceHanSansK-VF.otf[wght=800]"  % axis names & values
```

Both arguments before and after '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, or names and values for axis feature, of a variable font.

N.B. Regrettably we have some bug in processing not a few glyphs in cmr10.pfb and its family (or maybe other) Type1 fonts.[12] If that happens, consider using glyph operator instead of mplibglyph.

[10] Users can use the withmplibcolors macro instead of fillcolor and drawcolor options. See § 1.2.3 on this macro.
[11] But this limitation is now lifted by the introduction of withshadingmethod. See above § 1.2.6.
[12] The bug seems to be fixed in font-cff.lmt contained in ConTEXt mkxl, but current **luaotfload** is based on font-

1.2.10 mplibdrawglyph ..., mplibstrokeglyph ..., mplibfillandstrokeglyph ...

As the structure of the picture returned by mplibglyph is quite similar to the result of glyph primitive, 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.

N.B. 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 additionally declare withpostscript "evenodd" to the last path.

N.B. By the way, when you want fill-and-stroke effect, issueing filldraw command to the last path will not always produce what you want: in such cases, you have to issue the command draw ⟨*the last path*⟩ withpostscript "both" (or "eoboth" to apply even-odd rule).[13]

As this could be somewhat annoying to users, luamplib v2.38.0 or later provides the following commands as well: mplibfillandstrokeglyph ⟨*picture*⟩, mplibstrokeglyph ⟨*picture*⟩, and mplibfillglyph ⟨*picture*⟩, the last one being a synonym of mplibdrawglyph command.

An example:

```
mplibfillandstrokeglyph
  mplibglyph "R" of \fontid\font scaled 1/12
  withpen pencircle scaled 1
  withmplibcolors ("orange", 2/3red) ;
```

1.2.11 mpliboutlinetext (...)

As said before at § 1.1.3, luamplib provides the METAPOST operator mpliboutlinetext (⟨*string*⟩) which mimicks *metafun*'s outlinetext, but with some enhancements including the support for right-to-left writing direction. The syntax is the same as that of *metafun*: see the *metafun* documentation § 8.7 (texdoc metafun).

A simple example:

```
draw mpliboutlinetext.b ("$\sqrt{2+\alpha}$")
  (withcolor \mpcolor{red!50})
  (withpen pencircle scaled .25 withcolor 2/3red)
  scaled 3 ;
```

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 outline paths of 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.

---

cff.lua from ConTEXt mkiv. As you see, mplibglyph operator requires luaotfload package loaded, which however is done automatically by LATEX format.

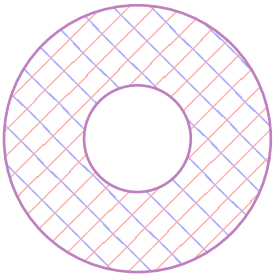
[13] *metafun* provides macros nofill, eofill, fillup, eofillup etc. (see *metafun* manual § 2.11), which luamplib with *plain* format does not provide currently.

1.2.12 `\mppattern{...} ... \endmppattern`, `... withmppattern ...`

TEX macros `\mppattern{`⟨*name*⟩`}` ... `\endmppattern` define a tiling pattern cell associated with the ⟨*name*⟩. METAPOST command `withmppattern`, the syntax being ⟨*cyclic path*⟩ | ⟨*textual picture*⟩ `withmppattern` ⟨*string*⟩, will fill the given path or text with the tiling pattern cell of the ⟨*name*⟩ by replicating it horizontally and vertically.[14] As said before at § 1.2.6, the *textual picture* here means any text typeset by TEX, mostly the result of the `btex` command (and its derivatives) or the `infont` operator.

An example:

```
\mppattern{mypatt}              % or \begin{mppattern}{mypatt}
  [                             % options: see below
    xstep = 10,
    ystep = 7,
    matrix = "rotated 45",      % or "0.7 0.7 -0.7 0.7" or {0.7, 0.7, -0.7, 0.7}
  ]
  \mpfig                        % or any other TeX code
    draw (up--down) scaled 5
      withcolor 2/3[blue,white] ;
    draw (left--right) scaled 5
      withcolor 2/3[red,white] ;
  \endmpfig
\endmppattern                   % or \end{mppattern}

\mpfig
  mplibdrawglyph image(
      draw fullcircle scaled 100;
      draw reverse fullcircle scaled 40;
    )
    withmppattern "mypatt"
    withpen pencircle scaled 1
    withcolor \mpcolor{red!50!blue!50} ;
\endmpfig
```



The available options, actually elements of a Lua *table*, are listed in Table 1. For the sake of convenience, the width and height values of the tiling pattern cell 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 the 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 effect such as transparency in a pattern cell, `resources` option is needed: for instance, `resources="/ExtGState <</MyObj 5 0 R>>"`. However, as **luamplib** automatically includes the resources of the current page, this option is not needed in most cases.

[14] `withpattern` is an operator virtually the same as `withmppattern`, but the former forces a METAPOST picture. Therefore you cannot but use `draw` command with `withpattern` operator. On the other hand, ⟨*cyclic path*⟩ `withmppattern` ⟨*string*⟩ works as intended only with `fill` or `filldraw` command.

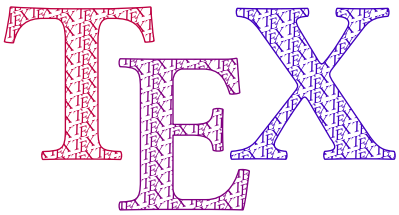
Table 1: options for \mppattern

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

Option colored=false (or coloured=false) will generate an uncolored pattern cell which shall have no color at all (i.e. withoutcolor command is needed for METAPOST code).[15] Uncolored pattern will be painted later by the color of a METAPOST object. An example:

```
\begin{mppattern}{pattnocolor}
  [
    colored = false,
    matrix = "slanted .3 rotated 15",
  ]
  \tiny\TeX
\end{mppattern}

\begin{mplibcode}
  beginfig(1)
    picture tex;
    tex = mpliboutlinetext ("\bfseries \TeX");
    for i=1 upto mpliboutlinenum:
      mplibfillandstrokeglyph mpliboutlinepic[i]
        scaled 8
        withmppattern "pattnocolor"
        withpen pencircle scaled 1/2
        withcolor (i/4)[red,blue]      % paints the pattern
        ;
    endfor
  endfig;
\end{mplibcode}
```



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

```
\begin{mplibcode}
  beginfig(2)
    draw mplibgraphictext "\bfseries\TeX"
```

[15] When using DVI mode, -c option might be needed to the dvipdfmx command.

```
      fakebold 1/2
      rotated 10 scaled 8
      withmppattern "pattnocolor"
      withmplibcolors (
        2/3[red,white],              % paints the pattern
        2/3 red
      ) ;
  endfig;
\end{mplibcode}
```

1.2.13 ... asgroup ...

As said before at § 1.1.3, transparency group is available with *plain* as well as *metafun*. 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 syntax is basically the same as *metafun*'s: ⟨*picture*⟩ | ⟨*path*⟩ `asgroup` ⟨*string*⟩, the latter being `""` | `"isolated"` | `"knockout"` | `"isolated,knockout"` | `"off"`, which will return a METAPOST picture. The additional features provided by **luamplib** are:

- As shown, in addition to those arguments mimicking *metafun*'s, we allow another argument at the right-hand side: `asgroup "off"` will produce an ordinary *form XObject* rather than a transparency group XObject. On the contrary, `asgroup ""` (empty string) will produce a transparency group in which both of the PDF keys `/I` and `/K` are `false`.

- 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.

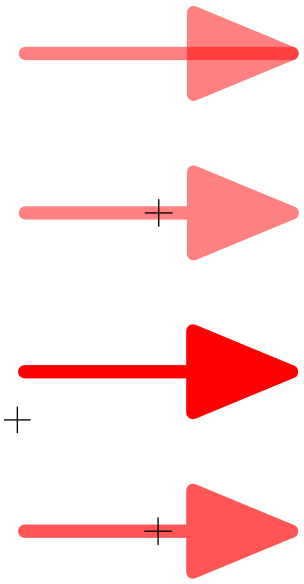
\usemplibgroup{⟨***name***⟩**}** is a TEX command to reuse a transparency group of the name once used. Note that the position of the group will be origin-based: in other words, lower-left corner of the bounding box will be shifted to the origin.

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 effect of transparency group and the difference between the TEX and METAPOST commands:

```
\mpfig
  picture pic;
  pic = image(drawarrow (left--right) scaled 5 withcolor red) scaled 10 ;
  draw pic
    asgroup "off"
    withtransparency (1, 1/2) ;
\endmpfig

\mpfig
  draw pic
    asgroup ""
    withgroupname "mygroup"
    withtransparency (1, 1/2) ;
  draw (left--right) scaled 5 ;
  draw (up--down) scaled 5 ;
\endmpfig

\noindent
\clap{\vrule width 10bp height .25bp depth .25bp}%
\clap{\vrule width .5bp height 5bp depth 5bp}%
\usemplibgroup{mygroup}

\mpfig
  usemplibgroup "mygroup"
    withtransparency (1, 2/3) ;
  draw (left--right) scaled 5 ;
  draw (up--down) scaled 5 ;
\endmpfig
```

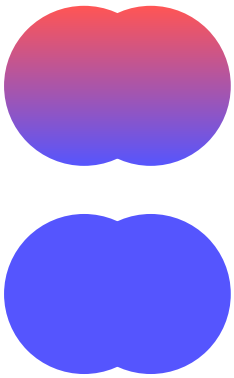


Also note that normally the transparency groups are not affected by outer color commands. However, if you have made the original transparency group using `withoutcolor` command, colors will have effects on the uncolored objects in the group.

N.B. When you give shading effect upon a *textual picture* (ie. non-path object) inside or outside a transparency group, currently many of the PDF renderers, including Mac OS Preview and Foxit Editor, do not interprete PDF coordinates properly. If that happens, consider using other PDF viewer such as Adobe Acrobat. An example:

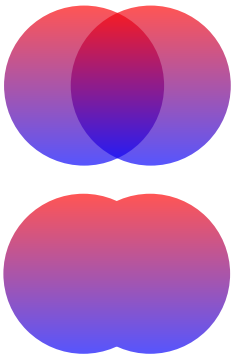
```
\mpfig*
  picture pic[];
  pic1 = image(
      fill fullcircle scaled 60 withoutcolor;
      fill fullcircle scaled 60 shifted 25right withoutcolor;
    ) ;
  pic2 = image(
    draw pic1
      withshadingmethod "linear"
```

```
      withshadingvector (2, 1)
      withshadingcolors (red, blue)
  ) ;
\endmpfig

\mpfig                              % shading inside group
  draw pic2
    asgroup ""
    withtransparency (1, 2/3) ;
\endmpfig

\mpfig                              % shading outside group
  draw pic1
    asgroup ""
    withshadingmethod "linear"
    withshadingvector (2, 1)
    withshadingcolors (red, blue)
    withtransparency (1, 2/3) ;
\endmpfig
```



After some experiment, however, it turned out that the best way to get picture shading with transparency group is to give shading effect within an ordinary form XObject and then wrap it in a transparency group XObject. Most PDF renderers do render it properly:

```
\mpfig                              % shading within ordinary xobject
  draw pic2
    asgroup "off"
    withgroupname "myshading"
    withtransparency (1, 2/3) ;
\endmpfig

\mpfig                              % wrap in a transparency group
  usemplibgroup "myshading"
    asgroup ""
    withtransparency (1, 2/3) ;
\endmpfig
```



or preferably (see next subsection § 1.2.14 on \mplibgroup):

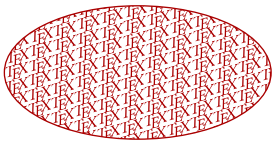
```
\mplibgroup{myshading}[asgroup="off"]           % or no option
  \mpfig
    draw pic2 ;
  \endmpfig
\endmplibgroup

\mpfig
  usemplibgroup "myshading"
    asgroup ""
    withtransparency (1, 2/3) ;
\endmpfig
```

1.2.14 \mplibgroup{...} ... \endmplibgroup

These TEX macros are described here in this subsection, as they are deeply related to the asgroup operator described just above at § 1.2.13. Users can define a transparency group or an ordinary *form XObject* with these macros from TEX side. The syntax is similar to the \mppattern command (see above § 1.2.12).

An example:[16]

```
\mplibgroup{texpatt1}                    % or \begin{mplibgroup}{texpatt1}
  [                                      % options: see below
    asgroup="off",
  ]
  \mpfig                                 % or any other TeX code
    filldraw fullcircle xscaled 100 yscaled 50
      withmppattern "pattnocolor"
      withcolor 2/3 red ;
  \endmpfig
\endmplibgroup                           % or \end{mplibgroup}

\mplibgroup{texpatt2}
  [
    asgroup="",
  ]
  \usemplibgroup{texpatt1}
\endmplibgroup

\usemplibgroup{texpatt2}

\mpfig
  usemplibgroup "texpatt2"
    rotated -15 scaled 1.2
    withtransparency (1, 2/3) ;
\endmpfig
```



Availabe options 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 is not given or is given as "off", an ordinary 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, just like the first figure in the example above at § 1.2.13.

As for the option asgroup="masking", see the next subsection § 1.2.15.

With colorspace option, which is no-op for ordinary form XObject, users can specify the color space of a transparency group. For instance, the mplibgroup will be painted in grayscale model when colorspace="/DeviceGray" is given to an *isolated* transparency group.[17]

[16] Note that, here again, within a transparency group XObject a tiling pattern is encapsulated in an inner, ordinary XObject. This annoyance is especially for Mac OS Preview which misapplies the transformation matrix to tiling or shading patterns directly wrapped in a transparency group. Most other PDF renderers seem to behave properly even with single wrapping in a transparency group.

[17] Note that some PDF renderers such as Mac OS Preview or Firefox do not render properly this option.

Table 2: options for `\mplibgroup`

Key	Value Type	Explanation
`asgroup`	*string*	`""`, `"isolated"`, `"knockout"`, `"isolated,knockout"`, `"off"` or `"masking"`
`colorspace`	*string*	`/CS` entry to a transparency group, such as `"/DeviceGray"`
`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

As shown, you can reuse the mplibgroup using the TEX command `\usemplibgroup` or the METAPOST command `usemplibgroup`. The behavior of these commands is the same as that described above at § 1.2.13, excepting that the mplibgroup made by TEX code (not by METAPOST code) respects original height and depth.

1.2.15 ... withmaskinggroup ...

Using this command, the mplibgroup (see above § 1.2.14) generated by the option `asgroup="masking"` (see Table 2) can be utilized as a masking transparency group upon a picture or a path object. The syntax is ⟨*picture*⟩ | ⟨*path*⟩ `withmaskinggroup` ⟨*string*⟩, the latter being the name of a pre-defined masking group.

Basically, the masking group should be prepared in *grayscale* color model: the area painted with `1` (≈ white: full luminosity) will preserve the full color of the object; the area painted with `0` (≈ black: zero luminosity) will force full transparency, masking it invisibly.[18]

By default, the background color of a masking group is `0` (≈ black), which you can change by this macro:

withmaskingbgcolor ⟨***color expr***⟩ sets the background color of the masking group. `0` denotes full transparency (masking invisibly); `1`, full color.[19]

An example:

```
\mpfig*
  picture pic;
  pic = image(
      fill fullcircle scaled 80 withcolor blue ;
      fill fullcircle scaled 80 shifted (25,0) withcolor green ;
      fill fullcircle scaled 80 shifted (50,0) withcolor red ;
    );
\endmpfig
```

[18] In fact, colors in other color models are also allowed (such as `white`, `black`, `red`, `green`, `blue`). But they will be converted to grayscale model by the PDF renderer, so that `"/DeviceGray"` is the default value of `colorspace` option to a masking transparency group (see Table 2 at § 1.2.14).

[19] Color expressions in rgb or cmyk model are also allowed, in accordance with the option `colorspace="/DeviceRGB"` or `colorspace="/DeviceCMYK"` given to the masking transparency group. This however we do not recommend as the luminosity is difficult to understand intuitively.

```
\mplibgroup{mymask}[asgroup="masking"]
  \mpfig
    label(TEX "\sffamily\bfseries\scshape\Huge Meta" scaled 2, center pic)
      withcolor 1 ;
  \endmpfig
\endmplibgroup

\mpfig
  fill bbox pic
    withshadingmethod "linear"
    withshadingcolors (red, blue) ;
  draw pic
    withmaskinggroup "mymask"
    withmaskingbgcolor 1/10
    withtransparency (1, 0.8) ;
\endmpfig
```

1.2.16 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 installed.

1.2.17 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 installed.

1.3 Lua

1.3.1 runscript ...

A good many METAPOST macros described in this documentation have been implemented using the primitive runscript. With 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 data 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.

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 example will print `false`, `3.0`, `MetaPost` and the knots and the cyclicity of the path `unitsquare`.

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

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

Of course, this sort of Lua code can also be run inside METAPOST code using `runscript` command. Again, of course you can access a METAPOST variable using your own TEX macro. For example:

```
\def\mpnumeric#1#2{\directlua{
  tex.sprint(tostring(luamplib.instances["#1"]:get_numeric"#2"))
}}
\mpnumeric{myinstance}{n}\relax
```

3.0

1.3.3 Lua function `luamplib.process_mplibcode`

Users can run 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 affect the process of `process_mplibcode`.

Table 3: elements in luamplib table (partial)

Key	Type	Related TEX macro	Cf.
codeinherit	*boolean*	\mplibcodeinherit	§ 1.1.8
everyendmplib	*table*	\everyendmplib	§ 1.1.2
everymplib	*table*	\everymplib	§ 1.1.2
getcachedir	*function* (⟨*string*⟩)	\mplibcachedir	§ 1.1.15
globaltextext	*boolean*	\mplibglobaltextext	§ 1.1.9
legacyverbatimtex	*boolean*	\mpliblegacybehavior	§ 1.1.6
noneedtoreplace	*table*	\mplibmakenocache	§ 1.1.15
numbersystem	*string*	\mplibnumbersystem	§ 1.1.4
setformat	*function* (⟨*string*⟩)	\mplibsetformat	§ 1.1.3
showlog	*boolean*	\mplibshowlog	§ 1.1.5
textextlabel	*boolean*	\mplibtextextlabel	§ 1.1.7
verbatiminput	*boolean*	\mplibverbatim	§ 1.1.11

1.3.4 Lua function `luamplib.registerpattern`

This is the Lua interface for \mppattern ... \endmppattern described above at § 1.2.12.

```
luamplib.registerpattern (<number> box register, <string> pattern name, <table> options)
```

The first argument is the register of a box containing a pattern cell, which should be prepared in advance by the user. For instance, \setbox0=\hbox{\tiny\TeX}, or corresponding Lua code using tex.setbox function; then the argument should be 0.[20]

As for the third argument, see above Table 1. The argument cannot be absent, but can be an empty table, i.e. { }.

1.3.5 Lua function `luamplib.registergroup`

This is the Lua interface for \mplibgroup ... \endmplibgroup described above at § 1.2.14.

```
luamplib.registergroup (<number> box register, <string> group name, <table> options)
```

The first argument is the register of a box prepared in advance by the user. When the contents of the box have been generated from TEX (not METAPOST) code, please make sure that both of the TEX macros 'MPllx' and 'MPlly' are defined as '0' before invoking the Lua function.[21]

As for the third argument, see above Table 2. The argument cannot be absent, but can be an empty table, i.e. { }.

Reusing an mplibgroup, \usemplibgroup{⟨*name*⟩}, is basically the same as running the TEX macro 'luamplib.group.⟨*name*⟩'. If you need the boxresource index, inspect this macro using token.get_macro function.

[20] In DVI mode, TEX macro 'mplibpatternname' should be set as ⟨*pattern name*⟩ before preparing the box, if shading pattern (ie. shading on picture) is used in the pattern cell.

[21] In DVI mode, TEX macro 'mplibgroupname' also should be set as ⟨*group name*⟩ before preparing the box, if shading pattern (ie. shading on picture) is used in the mplibgroup.

2 Implementation

2.1 Lua module

```
1
2 luatexbase.provides_module {
3   name          = "luamplib",
4   version       = "2.41.1",
5   date          = "2026/05/05",
6   description   = "Lua package to typeset Metapost with LuaTeX's MPLib.",
7 }
8
```

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

```
9 luamplib          = luamplib or { }
10 local luamplib    = luamplib
11
12 local format, abs = string.format, math.abs
13
```

Use our own function for warn/info/err.

```
14 local function termorlog (target, text, kind)
15   if text then
16     local mod, write, append = "luamplib", texio.write_nl, texio.write
17     kind = kind
18         or target == "term" and "Warning (more info in the log)"
19         or target == "log" and "Info"
20         or target == "term and log" and "Warning"
21         or "Error"
22     target = kind == "Error" and "term and log" or target
23     local t = text:explode"\n+"
24     write(target, format("Module %s %s:", mod, kind))
25     if #t == 1 then
26       append(target, format(" %s", t[1]))
27     else
28       for _,line in ipairs(t) do
29         write(target, line)
30       end
31       write(target, format("(%s)     ", mod))
32     end
33     append(target, format(" on input line %s", tex.inputlineno))
34     write(target, "")
35     if kind == "Error" then error() end
36   end
37 end
38 local function warn (...) -- beware '%' symbol
39   termorlog("term and log", select("#",...) > 1 and format(...) or ...)
40 end
41 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

```

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.

First of all, determine the directory to store cache files.

```
local cachedir
local function outputdir ()
  if lfstouch then
    for i,v in ipairs{'TEXMFVAR','TEXMF_OUTPUT_DIRECTORY','.','TEXMFOUTPUT'} do
      local var = i == 3 and v or kpse.var_value(v)
      if var and var ~= "" then
        for _,vv in ipairs(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
            cachedir = dir; return cachedir
          end
        end
      end
    end
  end
  cachedir = "."; return cachedir
end
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

```

Pattern formats to replace btex 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

```

Function luamplib.finder

```
local currenttime = os.time()
do
  local luamplibtime = lfsattributes(kpse.find_file"luamplib.lua", "modification")
```

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
  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] and not noneedtoreplace["*.mp"] 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
end

```

For the main function: process

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 process
do
  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
```

```
265           termorlog("term", first)
266           termorlog("log", log, "Warning")
267         else
268           warn(log)
269         end
270         if result.status > 1 then
271           err(e or "see above messages")
272         end
273       elseif prevlog then
274         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.

```
275         local show = log:match"\n>>? .+"
276         if show then
277           termorlog("term", show, "Info (more info in the log)")
278           info(log)
279         elseif luamplib.showlog and log:find"%g" then
280           info(log)
281         end
282       end
283       return log
284     end
285   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.

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

Make use of `make_text` and `run_script`. And we provide `numbersystem` option since v2.4. See https://github.com/lualatex/luamplib/issues/21.

```
291       make_text   = luamplib.maketext,
292       run_script  = luamplib.runscript,
293       math_mode   = luamplib.numbersystem,
294       job_name    = tex.jobname,
295       random_seed = math.random(4095),
296       utf8_mode   = true,
297       extensions  = 1,
298     }
```

Append our own METAPOST preamble to the preamble loading plain/metafun format.

```
299     local preamble = tableconcat{
300       format(luamplib.preambles.preamble, replacesuffix(name,"mp")),
301       luamplib.preambles.mplibcode,
302       luamplib.legacyverbatimtex and luamplib.preambles.legacyverbatimtex or "",
303       luamplib.textextlabel and luamplib.preambles.textextlabel or "",
304     }
```

```
  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}.

```
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),
      tostring(tex.currentgrouplevel), -- address #63
    }
    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
```

```
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
```

For conversion of sp to bp.

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

```

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 }
local process_tex_text
do
  local textext_fmt = 'image(addto currentpicture doublepath unitsquare \z
    xscaled %f yscaled %f shifted (0,-%f) \z
    withprescript "mplibtexboxid=%i:%f:%f")'
  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
      if str:find"^%[taggingoff%]" then
        str = str:gsub("^%[taggingoff%]%s*","")
        run_tex_code(format("\\luamplibnotagtextboxset{%i}{%s\\setbox%i\\hbox{%s}}",
                            tex_box_id, global, tex_box_id, str))
      else
```

```
390         run_tex_code(format("\\luamplibtagtextboxset{%i}{%s\\setbox%i\\hbox{%s}}",
391                             tex_box_id, global, tex_box_id, str))
392       end
393       local box = texgetbox(tex_box_id)
394       local wd  = box.width  / factor
395       local ht  = box.height / factor
396       local dp  = box.depth  / factor
397       return textext_fmt:format(wd, ht+dp, dp, tex_box_id, wd, ht+dp)
398     end
399     return ""
400   end
401 end
402
```

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.

```
403 if is_defined'color_select:n' then
404   run_tex_code{
405     "\\newcatcodetable\\luamplibcctabexplat",
406     "\\begingroup",
407     "\\catcode`@=11 ",
408     "\\catcode`_=11 ",
409     "\\catcode`:=11 ",
410     "\\savecatcodetable\\luamplibcctabexplat",
411     "\\endgroup",
412   }
413 end
414 local ccexplat = luatexbase.registernumber"luamplibcctabexplat"
415
416 local process_color, process_mplibcolor
```

A common function for color functions

```
417 local function colorsplit (res)
418   local t, tt = { }, res:gsub("[%[%]]","",2):explode()
419   local be = tt[1]:find"^%d" and 1 or 2
420   for i=be, #tt do
421     if not tonumber(tt[i]) then break end
422     t[#t+1] = tt[i]
423   end
424   if #t == 0 then -- named color in DVI mode with no DocumentMetadata
425     run_tex_code{"\\extractcolorspecs{", tt[3], "}\\mplibtmpa\\mplibtmpb"}
426     t = get_macro"mplibtmpb":explode","
427   end
428   return t
429 end
430 do
431   local colfmt = ccexplat and "l3color" or "xcolor"
432   local mplibcolorfmt = {
433     xcolor = tableconcat{
434       [[\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]],
  },
}
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 then
      if t:find"^hsb" or not t:find"%d" then
        t = "color push " .. t
      elseif not t:find"^pdf" then
        t = t:gsub("%a+ (.+)","pdf:bc [%1]")
      end
    end
    return format('1 withprescript "mpliboverridecolor=%s"', t)
  end
  return ""
end
function process_mplibcolor(str)
  local res = process_color(str)
  if res:find" cs " or res:find"@pdf.obj" or res:find"color push" then return res end
  res = colorsplit(res:match'"mpliboverridecolor=(.+)"')
  return format("(%s)", tableconcat(res, ","))
```

```
  end
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 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
```

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

```

Now `luamplib.runscript`

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

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
end
```

luamplib.maketext

```
luamplib.legacyverbatimtex = true
do
```

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
  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
end

```

luamplib's METAPOST color operators

```
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
```

luamplib.fillandstrokecolor

```
do
  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 = col:explode()
```

```
      local b = filldraw == "fill" and 1 or #t/2+1
      local e = b == 1 and #t/2 or #t
      return tableconcat(t," ", b, e)
    end
    if col:find"@pdf.obj" then
      return col:gsub("pdf:bc%s*","",1)
    else
      return format("[%s]", tableconcat(colorsplit(col)," "))
    end
  end
  function luamplib.fillandstrokecolor (fill, stroke)
    fill   = graphictextcolor(fill, "fill")
    stroke = graphictextcolor(stroke, "stroke")
    local bc = pdfmode and "" or "pdf:bc "
    return format('withprescript "mpliboverridecolor=%s%s %s"', bc, fill, stroke)
  end
end
```

Remove trailing zeros for smaller PDF

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

common function for mplibgraphictext and mpliboutlinetext

```
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
```

luamplib's mplibgraphictext operator

```
do
  if not math.round then
    function math.round(x) return x < 0 and -math.floor(-x + 0.5) or math.floor(x + 0.5) end
  end
  local emboldenfonts = { }
  local function roundupwidth (f, fb)
    local wd = math.round(f.size * fb / factor * 10)
    if wd == 0 and fb ~= 0 then
      wd = 1
    end
    emboldenfonts.width = wd
```

```
  return wd
end
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 = roundupwidth(font.getcopy(f or font.current()), fakebold)
  end
  return width
end
local function getrulewhatsit (line, wd, ht, dp)
  line, wd, ht, dp = line/1000, wd/factor, ht/factor, dp/factor
  line = line == 0 and "" or ("%f w"):format(line)
  local pl
  local fmt = "q %s %f %f %f %f re B Q"
  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) :gsub(decimals,rmzeros)
  local ss = node.new"glue"
  node.setglue(ss, 0, 65536, 65536, 2, 2)
  pl.next = ss
  return pl
end
```

copying attributes of rule/glue node to improve tagging of mplibgraphictext

```
local tag_update_attrs
if is_defined"ver@tagpdf.sty" then
  tag_update_attrs = function (n, curr)
    while n do
      n.attr = curr.attr
      if n.head then
        tag_update_attrs(n.head, curr)
      end
```

```
774         n = node.getnext(n)
775       end
776     end
777   else
778     tag_update_attrs = function() end
779   end
780   local function embolden (box, curr, fakebold)
781     local head = curr
782     while curr do
783       if curr.head then
784         curr.head = embolden(curr, curr.head, fakebold)
785       elseif curr.replace then
786         curr.replace = embolden(box, curr.replace, fakebold)
787       elseif curr.leader then
788         if curr.leader.head then
789           curr.leader.head = embolden(curr.leader, curr.leader.head, fakebold)
790         elseif curr.leader.id == node.id"rule" then
791           local glue = node.effective_glue(curr, box)
792           local line = getemboldenwidth(curr, fakebold)
793           local wd,ht,dp = getrulemetric(box, curr.leader)
794           if box.id == node.id"hlist" then
795             wd = glue
796           else
797             ht, dp = 0, glue
798           end
799           local pl = getrulewhatsit(line, wd, ht, dp)
800           local pack = box.id == node.id"hlist" and node.hpack or node.vpack
801           local list = pack(pl, glue, "exactly")
802           tag_update_attrs(list,curr)
803           head = node.insert_after(head, curr, list)
804           head, curr = node.remove(head, curr)
805         end
806       elseif curr.id == node.id"rule" and curr.subtype == 0 then
807         local line = getemboldenwidth(curr, fakebold)
808         local wd,ht,dp = getrulemetric(box, curr)
809         if box.id == node.id"vlist" then
810           ht, dp = 0, ht+dp
811         end
812         local pl = getrulewhatsit(line, wd, ht, dp)
813         local list
814         if box.id == node.id"hlist" then
815           list = node.hpack(pl, wd, "exactly")
816         else
817           list = node.vpack(pl, ht+dp, "exactly")
818         end
819         tag_update_attrs(list,curr)
820         head = node.insert_after(head, curr, list)
821         head, curr = node.remove(head, curr)
822       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)
        local width = roundupwidth(ft, fakebold)
        if ft.format == "opentype" or ft.format == "truetype" then
          local name = ft.name:gsub('"','''):gsub(';$','')
          local t = name:gsub("^file:",""):gsub("^name:",""):gsub("^kpse:",""):gsub("^my:","")
          name = format('%s%sembolden=%s;',name, t:find":" and ";" or ":", fakebold)
          _, i = fonts.constructors.readanddefine(name,ft.size)
        elseif pdfmode then
          local ft = table.copy(ft)
          ft.mode, ft.width = 2, width
          i = font.define(ft)
        else
          goto skip_type1
        end
        emboldenfonts[key] = i
      end
      curr.font = i
    end
    ::skip_type1::
    curr = node.getnext(curr)
  end
  return head
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 colors = luamplib.fillandstrokecolor(fc, dc)
  return format('%s %s)', fmt, colors)
end
end
```

luamplib's mplibglyph operator

```
do
  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
```

```
870       r = r + 360
871     end
872     return r
873   end
874   local function turning (t)
875     local r, n = 0, #t
876     for i=1,2 do
877       tableinsert(t, t[i])
878     end
879     for i=1,n do
880       r = r + getangle(t[i], t[i+1], t[i+2])
881     end
882     return r/360
883   end
884   local function glyphimage(t, fmt)
885     local q, p, r, towarn = {{},{}}
886     local function closepath(dots)
887       tableinsert(p, format("%scycle", dots or "--"))
888       tableinsert(q[ turning(r) > 0 and 1 or 2 ], tableconcat(p))
889     end
890     for i,v in ipairs(t) do
891       local cmd = v[#v]
892       local nt = t[i+1]
893       local final = not nt or nt[#nt] ~= "l" and nt[#nt] ~= "c"
894       if cmd == "m" then
895         if final then towarn = true end
896         p = {format('(%s,%s)',v[1],v[2])}
897         r = {{x=v[1],y=v[2]}}
898       else
899         if cmd == "l" then
900           local pt = t[i-1]
901           if (final or pt and pt[#pt] == "m") and r[1].x == v[1] and r[1].y == v[2] then
902           else
903             tableinsert(p, format('--(%s,%s)',v[1],v[2]))
904             tableinsert(r, {x=v[1],y=v[2]})
905           end
906           if final then closepath() end
907         elseif cmd == "c" then
908           tableinsert(p, format('..controls(%s,%s)and(%s,%s)',v[1],v[2],v[3],v[4]))
909           if final and r[1].x == v[5] and r[1].y == v[6] then
910             closepath ".."
911           else
912             tableinsert(p, format('..(%s,%s)',v[5],v[6]))
913             tableinsert(r, {x=v[5],y=v[6]})
914             if final then closepath() end
915           end
916         elseif cmd == "path" or cmd == "move" then
917         else
918           return mperr"unknown operator"
```

```
      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)), towarn
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
      or fontdata.shared and fontdata.shared.features.axis
    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
```

```
968       end
969     end
970     if kind ~= "opentype" and kind ~= "truetype" then
971       f = fid and fid > 0 and tex.fontname(fid) or f
972       if kpse.find_file(f, "tfm") then
973         return format("glyph %s of %q", tonumber(c) or format("%q",c), f)
974       else
975         filename = kpse.find_file(f, "type1 fonts")
976         if filename then
977           kind = "type1" -- there's bug in processing cmr family
978         else
979           return mperr"font not found"
980         end
981       end
982     end
983     local time = lfsattributes(filename,"modification")

  local k = format("shapes_%s(%s)[%s]%s", filename, subfont or "", instance or "",
                   luaotfload and luaotfload.version or "")

984     local k = format("shapes_%s(%s)[%s]", filename, subfont or "", instance or "")
985     local h = format(string.rep('%02x', 256/8), string.byte(sha2.digest256(k), 1, -1))
986     local newname = format("%s/%s.lua", cachedir or outputdir(), h)
987     local newtime = lfsattributes(newname,"modification") or 0
988     if time == newtime then
989       shapedata = require(newname)
990     end
991     if not shapedata then
992       if fonts then
993         local handler = kind == "type1" and fonts.handlers.afm or fonts.handlers.otf
994         shapedata = handler.readers.loadshapes(filename,subfont,instance)
995       end
996       if not shapedata then return mperr"loadshapes() failed. luaotfload not loaded?" end
997       table.tofile(newname, shapedata, "return")
998       lfstouch(newname, time, time)
999     end
1000    local gid = tonumber(c)
1001    if not gid then
1002      local uni = utf8.codepoint(c)
1003      for i,v in pairs(shapedata.glyphs) do
1004        if c == v.name or uni == v.unicode then
1005          gid = i; break
1006        end
1007      end
1008    end
1009    if not gid then return mperr"cannot get GID (glyph id)" end
1010    local fac = 1000 / (shapedata.units or 1000)
1011    local t = shapedata.glyphs[gid]; t = t and t.segments
1012    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
    local result, towarn = glyphimage(t, shapedata.format or kind)
    if towarn then
      warn("mplibglyph %s not working properly. Use glyph instead", f)
    end
    return result
  end
end

```

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

```
do
  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
        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, yscale = scale * extend * expand, scale * squeeze
      dx = dx - (r2l and curr.width/factor*expand or 0)
      local xoff, yoff = (curr.xoffset or 0)/factor, (curr.yoffset or 0)/factor
      local xpos, ypos = dx + xshift + xoff, yshift + yoff
      local vertical = ""
      if ft.shared and (ft.shared.features.vert or ft.shared.features.vrt2) then
        if ft.shared.features.vertical then -- luatexko
          vertical = "rotated 90"
          local data = ft.characters[curr.char] or { }
          if ft.hb then
            local hoff, voff = (data.luatexko_hoff or 0)/factor, (data.luatexko_voff or 0)/factor
            local charraise = (ft.luatexko_charraise or 0)/factor
            xpos, ypos = xpos - voff + hoff - charraise, ypos + hoff + voff + charraise
```

```
1158            else
1159              local cmds = data.commands or { {0,0}, {0,0} }
1160              local voff, hoff = -cmds[1][2]/factor, cmds[2][2]/factor
1161              xpos, ypos = xpos + hoff, ypos + voff
1162            end
1163          elseif curr ~= box.head then -- luatexja
1164            vertical = "rotated 90"
1165            local en = ft.parameters.quad/factor/2
1166            xpos, ypos = xpos - xoff - yoff + en, ypos - yoff + xoff - en
1167          end
1168        end
1169        local image
1170        if ft.format == "opentype" or ft.format == "truetype" then
1171          image = luamplib.glyph(curr.font, gid)
1172        else
1173          local name, scale = ft.name, 1
1174          local vf = font.read_vf(name, ft.size)
1175          if vf and vf.characters[gid] then
1176            local cmds = vf.characters[gid].commands or {}
1177            for _,v in ipairs(cmds) do
1178              if v[1] == "char" then
1179                gid = v[2]
1180              elseif v[1] == "font" and vf.fonts[v[2]] then
1181                name  = vf.fonts[v[2]].name
1182                scale = vf.fonts[v[2]].size / ft.size
1183              end
1184            end
1185          end
1186          image = format("glyph %s of %q scaled %f", gid, name, scale)
1187        end
1188        res[#res+1] = format("mpliboutlinepic[%i]:=%s xscaled %f yscaled %f slanted %f %s shifted (%f,%f);",
1189                             #res+1, image, xscale, yscale, slant, vertical, xpos, ypos)
1190        dx = dx + (r2l and 0 or curr.width/factor*expand)
1191      elseif curr.replace then
1192        local width = node.dimensions(curr.replace)/factor
1193        dx = dx - (r2l and width or 0)
1194        res = outline_horz(res, box, curr.replace, xshift+dx, yshift, width)
1195        dx = dx + (r2l and 0 or width)
1196      elseif curr.id == node.id"rule" then
1197        local wd, ht, dp = getrulemetric(box, curr, true)
1198        if wd ~= 0 then
1199          local hd = ht + dp
1200          dx = dx - (r2l and wd or 0)
1201          if hd ~= 0 and curr.subtype == 0 then
1202            res[#res+1] = rulefmt:format(#res+1, wd, hd, xshift+dx+wd/2, yshift+(ht-dp)/2)
1203          end
1204          dx = dx + (r2l and 0 or wd)
1205        end
1206      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
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

```

METAPOST preambles

```
luamplib.preambles = {
  preamble = [[
boolean mplib ; mplib := true ;
let dump = endinput ;
let normalfontsize = fontsize;
input %s ;
]],
  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, strokecolor,
    fb, fc, dc, graphictextpic, alsoordoublepath;
  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; let strokecolor = drawcolor;
  def alsoordoublepath expr p = if picture p: also else: doublepath fi p enddef;
  addto graphictextpic alsoordoublepath (origin--cycle) rest; graphictextpic:=nullpicture;
  def fakebold  primary c = enddef;
  let fillcolor = fakebold; let drawcolor = fakebold;
  let withfillcolor = fillcolor; let withdrawcolor = drawcolor; let strokecolor = 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;
numeric luamplib_tmp_num_; luamplib_tmp_num_ = 0;
def mplibdrawglyph expr g =
  luamplib_tmp_num_ := 0;
  for item within g:
    fill pathpart item
    if incr luamplib_tmp_num_ < length g: withpostscript "collect"; fi
  endfor
enddef;
let mplibfillglyph = mplibdrawglyph;
def mplibstrokeglyph expr g =
  luamplib_tmp_num_ := 0;
  for item within g:
    draw pathpart item
    if incr luamplib_tmp_num_ < length g: withpostscript "collect"; fi
  endfor
enddef;
def mplibfillandstrokeglyph expr g =
  luamplib_tmp_num_ := 0;
  for item within g:
    draw pathpart item withpostscript
    if incr luamplib_tmp_num_ < length g: "collect"; else: "both" fi
  endfor
enddef;
def withmplibcolors (expr f, s) =
  runscript("return luamplib.fillandstrokecolor('" &
    if not string f: colordecimals fi f & "','" &
    if not string s: colordecimals fi s & "')")
enddef;
def withmplibopacities (expr a, f, s) =
  withprescript "tr_alternative=" & if numeric a: decimal fi a
  withprescript "tr_transparency=" & decimal f & ":" & decimal s
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 withmaskinggroup s =
  if picture p:
    image(
      draw p;
      draw center p withprescript "mplibfadestate=stop";
    )
  else:
    p withprescript "mplibfadestate=stop"
  fi
  withprescript "mplibfadetype=masking"
  withprescript "mplibmaskname=" & s
enddef;
def withmaskingbgcolor expr c =
  withprescript "mplibmaskingbgcolor=" & colordecimals c
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 withprescript "sh_in_xobj=yes";
    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("\luamplibtagasgroupput{"& s &"}{\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 or (length p > 1):
      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 withshadingcenters (expr a, b) =
  withprescript "sh_center_a=" & ddecimal a
  withprescript "sh_center_b=" & ddecimal b
  withshadingtransform "no"
  withshadingfactor 1
enddef;
let withshadingpoints = withshadingcenters;
def withshadingextend (expr a, b) =
  withprescript "sh_extend=" &
    if a: "true" else: "false" fi & " " &
    if b: "true" else: "false" fi
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 withshadingstroke expr a =
  withprescript "sh_stroking=" & a
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;
]],
}
```

process_mplibcode

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

```
luamplib.verbatiminput = false
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;")
  else
```

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. However, we do not expand btex ... etex, verbatimtex ... etex, and string expressions.

```
    local t = { } -- to store btex, verbatimtex, string
    data = data:gsub(btex_etex, function(str)
      t[#t+1] = str
      return format("btex \\unexpanded{!l!u!a!%s!m!p!l!} etex ",  #t) -- space
    end)
    :gsub(verbatimtex_etex, function(str)
      t[#t+1] = str
      return format("verbatimtex \\unexpanded{!l!u!a!%s!m!p!l!} etex;", #t) -- semicolon
    end)
    :gsub('"(.-)"', function(str)
      t[#t+1] = str
```

```
      return format('"\\unexpanded{!l!u!a!%s!m!p!l!}"', #t)
    end)
    :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("!l!u!a!(%d+)!m!p!l!", function(str) return t[tonumber(str)] or str end)
  end
  process(data, instancename)
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(...)` etc.

```
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

```

Attempt to address #189, the displacement issue of pdf link boxes.

```
    local matrix = format("%f %f %f %f", sx, rx, ry, sy) :gsub(decimals,rmzeros)
    put2output("\\mplibputtextbox{%i}{%f}{%f}{%s}", n, tx, ty, matrix)
  end
end

```

Colors

```
local do_preobj_CR
do
  local prev_override_color
  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
end

```

For transparency, shading, fading, and pattern

```
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.fallback_update_resources = function (name,res,obj)
    texsprint{"\\special{pdf:put ", obj, " <<", res, ">>}"}
    local tabname = format("%s_res",name)
    if not pdfetcs[tabname] then
      texsprint{"\\luamplibateveryshipout{\\special{pdf:put @resources <</", name, " ", obj, ">>}}"}
      pdfetcs[tabname] = { }
    end
    tableinsert(pdfetcs[tabname], res)
  end
end

```

Transparency

```
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 do_preobj_TR
do
  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",
  }
  function do_preobj_TR(object,prescript)
    if object.postscript == "collect" then return end
    local opaq = prescript and prescript.tr_transparency
    if not opaq then return end

    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 = opaq:explode":"
    for i,v in ipairs(opaq) do
      opaq[i] = format("%.3f", v) :gsub(decimals,rmzeros)
    end
    for i,v in ipairs{ {mode,opaq[1],opaq[2] or opaq[1]},{"Normal",1,1} } do
      os = format("<</BM/%s/ca %s/CA %s/AIS false>>",v[1],v[2],v[3])
      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,extend)
  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),
```

```
2157    } :gsub(decimals,rmzeros)
2158  else
2159    os = fun2fmt:format(domain,tableconcat(ca[1],' '),tableconcat(cb[1],' '))
2160      :gsub(decimals,rmzeros)
2161  end
2162  local objref = format(pdfetcs.resfmt, update_pdfobjs(os))
2163  os = tableconcat {
2164    format("<</ShadingType %i", shtype),
2165    format("/ColorSpace %s",    colorspace),
2166    format("/Function %s",      objref),
2167    format("/Coords[%s]",       coordinates),
2168    format("/Extend[%s]/AntiAlias true>>", extend or "true true")
2169  } :gsub(decimals,rmzeros)
2170  local on, new = update_pdfobjs(os)
2171  if new then
2172    local key, val = format("MPlibSh%s", on), format(pdfetcs.resfmt, on)
2173    if pdfmanagement then
2174      texsprint {
2175        "\\csname pdfmanagement_add:nnn\\endcsname{Page/Resources/Shading}{", key, "}{", val, "}"
2176      }
2177    else
2178      local res = format("/%s %s", key, val)
2179      pdfetcs.fallback_update_resources("Shading",res,"@MPlibSh")
2180    end
2181  end
2182  return on
2183 end
2184
2185 local do_preobj_SH
2186 do
2187  pdfetcs.clrspcs = setmetatable({ }, { __index = function(t,names)
2188    run_tex_code({
2189      [[\color_model_new:nnn]],
2190      format("{mplibcolorspace_%s}", names:gsub(",","_")),
2191      format("{DeviceN}{names={%s}}", names),
2192      [[\edef\mplib_@tempa{\pdf_object_ref_last:}]],
2193    }, ccexplat)
2194    local colorspace = get_macro'mplib_@tempa'
2195    t[names] = colorspace
2196    return colorspace
2197  end })
2198  local function color_normalize(ca,cb)
2199    if #cb == 1 then
2200      if #ca == 4 then
2201        cb[1], cb[2], cb[3], cb[4] = 0, 0, 0, 1-cb[1]
2202      else -- #ca = 3
2203        cb[1], cb[2], cb[3] = cb[1], cb[1], cb[1]
2204      end
2205    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
  function do_preobj_SH(object, prescript)
    local shade_no
    local sh_type = prescript and prescript.sh_type
    if not sh_type then return end

    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
```

```
2255  if prescript.mplib_spotcolor then
2256    ca, cb = { }, { }
2257    local names, pos, objref = { }, -1, ""
2258    local script = object.prescript:explode"\13+"
2259    for i=#script,1,-1 do
2260      if script[i]:find"mplib_spotcolor" then
2261        local t, name, value = script[i]:explode"="[2]:explode":"
2262        value, objref, name = t[1], t[2], t[3]
2263        if not names[name] then
2264          pos = pos+1
2265          names[name] = pos
2266          names[#names+1] = name
2267        end
2268        t = { }
2269        for j=1,names[name] do t[#t+1] = 0 end
2270        t[#t+1] = value
2271        tableinsert(#ca == #cb and ca or cb, t)
2272      end
2273    end
2274    for _,t in ipairs{ca,cb} do
2275      for _,tt in ipairs(t) do
2276        for i=1,#names-#tt do tt[#tt+1] = 0 end
2277      end
2278    end
2279    if #names == 1 then
2280      colorspace = objref
2281    else
2282      colorspace = pdfetcs.clrspcs[ tableconcat(names,",") ]
2283    end
2284  else
2285    local model = 0
2286    for _,t in ipairs{ca,cb} do
2287      for _,tt in ipairs(t) do
2288        model = model > #tt and model or #tt
2289      end
2290    end
2291    for _,t in ipairs{ca,cb} do
2292      for _,tt in ipairs(t) do
2293        if #tt < model then
2294          color_normalize(model == 4 and {1,1,1,1} or {1,1,1},tt)
2295        end
2296      end
2297    end
2298    colorspace = model == 4 and "/DeviceCMYK"
2299              or model == 3 and "/DeviceRGB"
2300              or model == 1 and "/DeviceGray"
2301              or err"unknown color model"
2302  end
2303  local extend = prescript.sh_extend
```

```
    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,extend)
    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,extend)
    else
      err"unknown shading type"
    end
    return shade_no, prescript.sh_stroking == "yes"
  end
end

```

Shading Patterns: we can apply shading to textual pictures as well as paths.

```
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")
    end
  end
end
if not pdfmode then
  pdfetcs.shadingpatterns = { }
  pdfetcs.shadingpatterninit_r, pdfetcs.shadingpatterninit_w = true, true
end
function luamplib.dolatelua (on, os, xobj)
  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 t = pdfetcs.shadingpatterns[on] or { }
```

```
    local shift = os == "group" and pdfetcs.tr_group.shifts[xobj]
               or os == "pattern" and pdfetcs.patterns[xobj].shifts
    if shift then
      h, v = -shift[1], -shift[2] -- engine bug in dvi mode?
    end
    if tonumber(h) ~= tonumber(t[1]) or tonumber(v) ~= tonumber(t[2]) then
      warn"Rerun to get correct shading pattern"
    end
    local name = format("%s/%s_shadingpatterns.aux", cachedir or outputdir(), tex.jobname)
    local init = pdfetcs.shadingpatterninit_w
    if init then pdfetcs.shadingpatterninit_w = nil end
    local f = ioopen(name, init and "w" or "a")
    if f then
      f:write(("%s %s %s\n"):format(on, h, v))
      f:close()
    else
      err"cannot write a file. check the cache dir path"
    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))
  if prescript.sh_in_xobj == "yes" then
    on = update_pdfobjs(("<<%s>>"):format(os))
    goto skip_latelua
  end
  on = update_pdfobjs()
  if pdfmode then
    put2output(tableconcat{ "\\latelua{ luamplib.dolatelua(",on,",[[",os,"]]) }" })
  else
    local xobj = is_defined"mplibgroupname" and {"group", get_macro"mplibgroupname"}
              or is_defined"mplibpatternname" and {"pattern", get_macro"mplibpatternname"}
    if xobj or not is_defined"RecordProperties" then -- in xobject or plain
      local init = pdfetcs.shadingpatterninit_r
      if init then
        pdfetcs.shadingpatterninit_r = nil
        local name = format("%s/%s_shadingpatterns.aux", cachedir or outputdir(), tex.jobname)
        local f = ioopen(name)
        if f then
          for line in f:lines() do
            local t = line:explode()
            pdfetcs.shadingpatterns[ tonumber(t[1]) ] = { t[2], t[3] }
          end
          f:close()
        end
      end
      local t = pdfetcs.shadingpatterns[on] or { 0, 0 }
```

```
      texsprint{ "\\special{pdf:put ", format(pdfetcs.resfmt, on),
                format(" <<%s/Matrix[1 0 0 1 %s %s]>>}", os, t[1], t[2]) }
      put2output("\\latelua{ luamplib.dolatelua(%s,%s) }", on,
                xobj and ("'%s',[[%s]]"):format(xobj[1], xobj[2]))
    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}
```

```
      put2output(tableconcat{
        "\\csname tex_savepos:D\\endcsname\\RecordProperties{luamplib/getpos/",on,"}{xpos,ypos}\z
        \\special{pdf:put ",format(pdfetcs.resfmt, 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
        ]>>}"
      })
    end
  end
  ::skip_latelua::
  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, ispattern)
  local t = { }
  if pdfmanagement then
    for _,v in ipairs {"ExtGState","ColorSpace","Pattern","Shading"} do
      local mytoks
      run_tex_code ({
        "\\mplibtmptoks\\expanded{{",
        "\\pdfdict_if_empty:nF{g__pdf_Core/Page/Resources/",v,"}",
        "{\\pdfdict_use:n{g__pdf_Core/Page/Resources/",v,"}}", "}}",
      },ccexplat)
      mytoks = texgettoks"mplibtmptoks"
      if not pdfmode then
        mytoks = mytoks:gsub("\\str_convert_pdfname:n%s*{(.-)}","%1") -- why not expanded?
      end
      mytoks = mytoks and mytoks:gsub("^%s*(.-)%s*$", "%1")
      if mytoks and mytoks ~= "" then
        t[#t+1] = ("/%s<<%s>>"):format(v, mytoks)
```

```
    end
  end
elseif is_defined(pdfetcs.pgfextgs) then
  run_tex_code"\\relax" -- flush tex.sprint queue
  if pdfmode then
    for k,v in pairs { ExtGState  = "pgf@sys@pgf@resource@list@extgs",
                       ColorSpace = "pgf@sys@pgf@resource@list@colorspaces",
                       Pattern    = "pgf@sys@pgf@resource@list@patterns", } do
      local res = (get_macro(v) or ""):gsub("^%s*(.-)%s*$","%1")
      if res ~= "" then
        t[#t+1] = ("/%s<<%s>>"):format(k, res )
      end
    end
  else
    local abc = get_macro"pgfutil@abc" or ""
    for k,v in pairs { ExtGState  = "@pgfextgs",
                       ColorSpace = "@pgfcolorspaces",
                       Pattern    = "@pgfpatterns", } do
      local tt = { }
      for vv in abc:gmatch( v .. "%s*(%b<>)" ) do
        tt[#tt+1] = vv:match("^<<%s*(.-)%s*>>$")
      end
      if #tt > 0 then
        t[#t+1] = ("/%s<<%s>>"):format(k, tableconcat(tt) )
      end
    end
  end
```

We still have to deal with Shading resources.

```
  if luatexbase.callbacktypes.finish_pdffile then
    if pdfetcs.Shading_res then
      t[#t+1] = ("/Shading<<%s>>"):format( tableconcat(pdfetcs.Shading_res) )
    end
  else
    local res = pdfetcs.getpageres()
    res = res and res:match"/Shading%s*%b<>"
    if res then
      t[#t+1] = res
    end
  end
else
  if ispattern and is_defined"TRP@list" then
```

We do not gather **transparent** package's `\TRP@list` as Acrobat glitches on tiling pattern plus masking group, so warn users and recommend `\DocumentMetadata`

```
    warn"transparent package is not fully functional without pdfmanagement code."
  end
  if luatexbase.callbacktypes.finish_pdffile then
    for _,v in ipairs {"ExtGState","ColorSpace","Pattern","Shading"} do
      local tt = pdfetcs[v.."_res"]
```

```
      if tt then
        t[#t+1] = ("/%s<<%s>>"):format(v, tableconcat(tt))
      end
    end
  else
    local res = pdfetcs.getpageres()
    if res then
      t[#t+1] = res
    end
  end
end
local result = tableconcat(t)
if optres ~= "" then
  for _,v in ipairs {"ExtGState","ColorSpace","Pattern","Shading"} do
    local res = optres:match("/"..v.."%s*%b<>")
    if res then
      if result:find("/"..v) then
        res = res:match("<<(.+)>>$")
        result = result:gsub("/"..v.."%s*<<", "%1"..res, 1)
      else
        result = result .. res
      end
    end
  end
end
return result
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 = gather_resources(optres, true) -- tiling pattern plus masking glitches with acrobat
  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 }
    patterns[name].shifts = { get_macro"MPllx", get_macro"MPlly" } -- for shading patterns above
  end
end

local do_preobj_PAT
do
  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
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
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 on, os, new
  if fd_type == "masking" then
    local mac = get_macro("luamplib.group."..prescript.mplibmaskname)
    on = mac:match(pdfmode and "%d+" or "{pdf:uxobj (.-)}")
    local bc = prescript.mplibmaskingbgcolor
    bc = bc and bc:gsub(":"," ")
    bc = bc and ("/BC[%s]"):format(bc):gsub(decimals,rmzeros) or ""
    os = format("<</SMask<</S/Luminosity/G %s%s>>>>",
                pdfmode and format(pdfetcs.resfmt, on) or on, bc)
  else
    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":"
    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 = format("<</SMask<</S/Luminosity/G %s>>>>", format(pdfetcs.resfmt, on))
  end
  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, trgroup.off = false, 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\luamplibtagasgroupset]]
  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
  if trgroup.off then
    grattr = ""
  else
    local on = update_pdfobjs(format("<</S/Transparency/I %s/K %s>>",
                                     trgroup.isolated, trgroup.knockout))
    grattr = format("/Group %s", pdfetcs.resfmt:format(on))
  end
  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",
      "\\luamplibtagasgroupput{",trgroup.name,"}{",
      [[\setbox\mplibscratchbox\hbox{\useboxresource\lastsavedboxresourceindex}]],
      [[\wd\mplibscratchbox 0pt\ht\mplibscratchbox 0pt\dp\mplibscratchbox 0pt]],
      [[\box\mplibscratchbox]],
      "}\\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, ">>}",
      "\\luamplibtagasgroupput{",trgroup.name,"}{",
      "\\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 is_mask = opts.asgroup and opts.asgroup:find"masking"
  local res = opts.resources or ""
  res = gather_resources(res)
  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
  local mpllx, mplly = get_macro'MPllx', get_macro'MPlly'
  if is_mask then
    local t = opts.matrix and opts.matrix:explode() or {1, 0, 0, 1, 0, 0}
    t[5], t[6] = t[5]+mpllx, t[6]+mplly
    opts.matrix = format("%f %f %f %f %f %f",tableunpack(t))
    mpllx, mplly = 0, 0
  end
  if opts.matrix then
    attr[#attr+1] = format("/Matrix[%s]", opts.matrix)
    grtype = opts.bbox and 4 or 1
  end
  if opts.asgroup and not opts.asgroup:find"off" then
    local t = { isolated = false, knockout = false, masking = false }
    for _,v in ipairs(opts.asgroup:explode",+") do t[v] = true end
    local on
    if t.masking then
      on = update_pdfobjs(format("<</S/Transparency/CS%s>>", opts.colorspace or "/DeviceGray"))
    else
      local cs = opts.colorspace and ("/CS%s"):format(opts.colorspace) or ""
      on = update_pdfobjs(format("<</S/Transparency%s/I %s/K %s>>", cs, t.isolated, t.knockout))
    end
    attr[#attr+1] = format("/Group %s", pdfetcs.resfmt:format(on))
  end
  local trgroup = pdfetcs.tr_group
  trgroup.shifts[name] = { mpllx, 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
```

luamplib.convert: flushing figures

```
do
  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
      if over:find"pdf:bc" then
        put2output"\\special{pdf:ec}"
      else
        put2output"\\special{color pop}"
      end
    end
```

```
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

```

Codes below to insert PDF lieterals are mostly from ConTEXt general, with small changes when needed.

```
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 then
    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 then
      return false
    end
  end
```

```
2915    return true
2916  end
2917
2918  local function flushnormalpath(path,open)
2919    local pth, ith
2920    for i=1,#path do
2921      pth = path[i]
2922      if not ith then
2923        pdf_literalcode("%f %f m",pth.x_coord,pth.y_coord)
2924      elseif curved(ith,pth) then
2925        pdf_literalcode("%f %f %f %f %f %f c",
2926          ith.right_x,ith.right_y,pth.left_x,pth.left_y,pth.x_coord,pth.y_coord)
2927      else
2928        pdf_literalcode("%f %f l",pth.x_coord,pth.y_coord)
2929      end
2930      ith = pth
2931    end
2932    if not open then
2933      local one = path[1]
2934      if curved(pth,one) then
2935        pdf_literalcode("%f %f %f %f %f %f c",
2936          pth.right_x,pth.right_y,one.left_x,one.left_y,one.x_coord,one.y_coord )
2937      else
2938        pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
2939      end
2940    elseif #path == 1 then -- special case .. draw point
2941      local one = path[1]
2942      pdf_literalcode("%f %f l",one.x_coord,one.y_coord)
2943    end
2944  end
2945
2946  local function flushconcatpath(path,open)
2947    pdf_literalcode("%f %f %f %f %f %f cm", sx, rx, ry, sy, tx ,ty)
2948    local pth, ith
2949    for i=1,#path do
2950      pth = path[i]
2951      if not ith then
2952        pdf_literalcode("%f %f m",concat(pth.x_coord,pth.y_coord))
2953      elseif curved(ith,pth) then
2954        local a, b = concat(ith.right_x,ith.right_y)
2955        local c, d = concat(pth.left_x,pth.left_y)
2956        pdf_literalcode("%f %f %f %f %f %f c",a,b,c,d,concat(pth.x_coord, pth.y_coord))
2957      else
2958        pdf_literalcode("%f %f l",concat(pth.x_coord, pth.y_coord))
2959      end
2960      ith = pth
2961    end
2962    if not open then
2963      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.

```
  local function flush (result,flusher)
    if result then
      local figures = result.fig
      if figures then
        for f=1, #figures do
          info("flushing figure %s",f)
          local figure = figures[f]
          local objects = figure:objects()
          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.

```
3002              local prescript     = object.prescript
3003              prescript = prescript and script2table(prescript) -- prescript is now a table
3004              local cr_over = do_preobj_CR(object,prescript) -- color
3005              local tr_opaq = do_preobj_TR(object,prescript) -- opacity
3006              local fading_ = do_preobj_FADE(object,prescript) -- fading
3007              local pattern_ = do_preobj_PAT(object,prescript) -- tiling pattern
3008              local shading_ = do_preobj_shading(object,prescript) -- shading pattern
3009              local trgroup = do_preobj_GRP(object,prescript) -- transparency group
3010              if prescript and prescript.mplibtexboxid then
3011                put_tex_boxes(object,prescript)
3012              elseif objecttype == "start_bounds" or objecttype == "stop_bounds" then --skip
3013              elseif objecttype == "start_clip" then
3014                local evenodd = not object.istext and object.postscript == "evenodd"
3015                start_pdf_code()
3016                flushnormalpath(object.path,false)
3017                pdf_literalcode(evenodd and "W* n" or "W n")
3018              elseif objecttype == "stop_clip" then
3019                stop_pdf_code()
3020                miterlimit, linecap, linejoin, dashed = -1, -1, -1, false
3021              elseif objecttype == "special" then
```

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

```
3022                if prescript and prescript.postmplibverbtex then
3023                  figcontents.post[#figcontents.post+1] = prescript.postmplibverbtex
3024                end
3025              elseif objecttype == "text" then
3026                local ot = object.transform -- 3,4,5,6,1,2
3027                start_pdf_code()
3028                pdf_literalcode("%f %f %f %f %f %f cm",ot[3],ot[4],ot[5],ot[6],ot[1],ot[2])
3029                pdf_textfigure(object.font,object.dsize,object.text,object.width,object.height,object.depth)
3030                stop_pdf_code()
3031              elseif not trgroup and fading_ ~= "stop" then
3032                local evenodd, collect, both = false, false, false
3033                local postscript = object.postscript
3034                if not object.istext then
3035                  if postscript == "evenodd" then
3036                    evenodd = true
3037                  elseif postscript == "collect" then
3038                    collect = true
3039                  elseif postscript == "both" then
3040                    both = true
3041                  elseif postscript == "eoboth" then
3042                    evenodd = true
3043                    both    = true
3044                  end
3045                end
3046                if collect then
3047                  if not savedpath then
```

```
3048                    savedpath = { object.path or false }
3049                    savedhtap = { object.htap or false }
3050                  else
3051                    savedpath[#savedpath+1] = object.path or false
3052                    savedhtap[#savedhtap+1] = object.htap or false
3053                  end
3054                else
```

Removed from ConTEXt general: color stuff.

```
3055                  local ml = object.miterlimit
3056                  if ml and ml ~= miterlimit then
3057                    miterlimit = ml
3058                    pdf_literalcode("%f M",ml)
3059                  end
3060                  local lj = object.linejoin
3061                  if lj and lj ~= linejoin then
3062                    linejoin = lj
3063                    pdf_literalcode("%i j",lj)
3064                  end
3065                  local lc = object.linecap
3066                  if lc and lc ~= linecap then
3067                    linecap = lc
3068                    pdf_literalcode("%i J",lc)
3069                  end
3070                  local dl = object.dash
3071                  if dl then
3072                    local d = format("[%s] %f d",tableconcat(dl.dashes or {}," "),dl.offset)
3073                    if d ~= dashed then
3074                      dashed = d
3075                      pdf_literalcode(dashed)
3076                    end
3077                  elseif dashed then
3078                    pdf_literalcode("[] 0 d")
3079                    dashed = false
3080                  end
3081                  local path = object.path
3082                  local transformed, penwidth = false, 1
3083                  local open = path and path[1].left_type and path[#path].right_type
3084                  local pen = object.pen
3085                  if pen then
3086                    if pen.type == 'elliptical' then
3087                      transformed, penwidth = pen_characteristics(object) -- boolean, value
3088                      pdf_literalcode("%f w",penwidth)
3089                      if objecttype == 'fill' then
3090                        objecttype = 'both'
3091                      end
3092                    else -- calculated by mplib itself
3093                      objecttype = 'fill'
3094                    end
```

```
end
```

Added : shading

```
local shade_no, shade_stroking = 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. Beware `q ... Q` scope.

```
                if shade_no then -- shading
                  pdf_literalcode("W%s %s /MPlibSh%s sh Q",
                    evenodd and "*" or "", shade_stroking and "s" or "n", 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.convert (result, flusher)
    flush(result, flusher)
    return true -- done
  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}
    [2026/05/05 v2.41.1 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. Plus address the issue #189. bdir 0 seems to be redundant, but no harm.

```
\ifnum\outputmode>0 %
  \def\mplibputtextbox#1#2#3#4{%
    \pdfextension save\relax
    \vbox to 0pt{\vss
      \hbox bdir0 to 0pt{\kern#2bp\pdfextension setmatrix{#4}\raise\dp#1\copy#1\hss}%
      \kern#3bp}%
    \pdfextension restore\relax}
\else
  \def\mplibputtextbox#1#2#3#4{%
    \special{pdf:btrans matrix #4 #2 #3}%
    \vbox to 0pt{\vss\hbox bdir0 to 0pt{\raise\dp#1\copy#1\hss}}%
    \special{pdf:etrans}}
\fi
```

use Transparency Group

```
\protected\def\usemplibgroup#1#{\usemplibgroupmain}
\def\usemplibgroupmain#1{%
  \prependtomplibbox\hbox dir TLT\bgroup
  \csname luamplib.group.#1\endcsname
  \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\MPllx{0}\def\MPlly{0}%
  \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
```

TEX macros for dimen/color

```
\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,\stop,}
\def\mplibdomakenocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdomakenocache
  \else
    \ifx\stop#1\else
      \directlua{luamplib.noneedtoreplace["#1.mp"]=true}%
      \expandafter\expandafter\expandafter\mplibdomakenocache
    \fi
  \fi
}
\def\mplibcancelnocache#1{\mplibdocancelnocache #1,\stop,}
\def\mplibdocancelnocache#1,{%
  \ifx\empty#1\empty
    \expandafter\mplibdocancelnocache
  \else
    \ifx\stop#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\luamplibtagtextboxset#1#2{#2}
\let\luamplibnotagtextboxset\luamplibtagtextboxset
\let\luamplibtagasgroupset\relax
\let\luamplibtagasgroupput\luamplibtagtextboxset
\ifcsname SuspendTagging\endcsname\else\endinput\fi
\ifcsname ver@tagpdf.sty\endcsname \else
  \ExplSyntaxOn
  \keys_define:nn{luamplib/tagging}
    {
      ,alt          .code:n = { }
      ,actualtext   .code:n = { }
      ,artifact     .code:n = { }
      ,text         .code:n = { }
      ,off          .code:n = { }
      ,tag          .code:n = { }
      ,adjust-BBox  .code:n = { }
      ,tagging-setup .code:n = { }
      ,instance     .code:n = { \tl_gset:Nn \currentmpinstancename {#1} }
      ,instancename .meta:n = { instance = {#1} }
      ,unknown      .code:n = { \tl_gset:NV \currentmpinstancename \l_keys_key_str }
```

```
    }
  \RenewDocumentCommand\mplibcode{O{}}
    {
      \tl_gclear:N \currentmpinstancename
      \keys_set:ne{luamplib/tagging}{#1}
      \mplibtmptoks{}\ltxdomplibcode
    }
  \cs_set_eq:NN \mplibalttext \use_none:n
  \cs_set_eq:NN \mplibactualtext \use_none:n
```

2025/12/05: `\begin{center}\mpfig ...\endmpfig\end{center}` raises an Error! as we issue `\everypar{}` before flushing literals out. It is related to `\partokencontext=2` recently introduced by LaTeX. Why we used `vbox` initially? where `hbox` seems to be sufficient. Anyway, among various solutions including `\partokencontext\z@`, `\let\par\@@par`, and `\endgraf`, we here attempt to address the issue by adding the following line, which LaTeX's `\everypar` should have done.

```
  \tl_put_left:Nn \mplibstoptoPDF \@newlistfalse
  \ExplSyntaxOff
  \endinput\fi
\ExplSyntaxOn
\tl_new:N \l__luamplib_tag_envname_tl
\tl_new:N \l__luamplib_tag_alt_tl
\tl_new:N \l__luamplib_tag_alt_dflt_tl
\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_bboxcorr_bool
\seq_new:N  \l__luamplib_tag_bboxcorr_seq
\tl_new:N \l__luamplib_tag_bbox_draw_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
\msg_new:nnn {luamplib}{figure-text-reuse}
{
  tex-text~box~#1~probably~is~incorrectly~tagged.~
  Reusing~a~box~in~text~mode~is~strongly~discouraged.~
  Check~the~resulting~PDF.
}
\msg_new:nnn {luamplib}{mplibgroup-text-mode}
{
  mplibgroup~'#1'~probably~is~incorrectly~tagged.~
  Using~mplibgroup~with~text~mode~is~not~recommended.~
  Check~the~resulting~PDF.
}
\msg_new:nnn{luamplib}{alt-text-missing}
{
  Alternate~text~for~#1~is~missing.~
  Using~the~default~value~'#2'~instead.
```

```
}
```

Sockets for tex-text boxes.

```
\socket_new:nn{tagsupport/luamplib/textext/set}{2}
\socket_new:nn{tagsupport/luamplib/textext/put}{2}
\socket_new_plug:nnn{tagsupport/luamplib/textext/set}{default}
{
```

TODO: we check text mode here. If we tag text boxes for all modes, we will get a lot of structure-has-no-parent warning; no good-looking, though it seems to be no harm.

```
  \bool_if:NTF \l__luamplib_tag_usetext_bool
  {
    \tag_mc_end_push:
    \tag_struct_begin:n{tag=NonStruct, stash, parent-tag=text}
    \cs_gset_nopar:cpe {luamplib.taggedbox.#1} {\tag_get:n{struct_num}}
```

TODO: We force an MC. Otherwise a and b in btex a x b etex are not tagged.

```
    \tag_mc_begin:n{tag=text}
    #2
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin_pop:n{}
  }
  {
    \tag_suspend:n{\luamplibtagtextboxset}
    #2
    \tag_resume:n{\luamplibtagtextboxset}
  }
}
\socket_new_plug:nnn{tagsupport/luamplib/textext/put}{default}
{
  \bool_lazy_and:nnTF
  { \l__luamplib_tag_usetext_bool }
  { \cs_if_free_p:c {luamplib.notaggedbox.#1} }
  {
    \tag_resume:n{\mplibputtextbox}
    \tag_mc_end:
    \cs_if_exist:cTF {luamplib.taggedbox.#1}
    {
      \exp_args:Nc \tag_struct_use_num:n {luamplib.taggedbox.#1}
      #2
      \cs_undefine:c {luamplib.taggedbox.#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
      #2
      \tag_mc_end:
```

```
    }
    \tag_mc_begin:n{artifact}
  }
  {
    \int_set:Nn \l_tmpa_int {#1}
    \tag_mc_reset_box:N \l_tmpa_int
    #2
  }
}
\socket_assign_plug:nn{tagsupport/luamplib/textext/set}{default}
\socket_assign_plug:nn{tagsupport/luamplib/textext/put}{default}
\cs_set_nopar:Npn \luamplibtagtextboxset
{
  \tag_socket_use:nnn{luamplib/textext/set}
}
```

For tex-text boxes starting with [taggingoff], which we will not tag at all. They will be just in the artifact MC-chunks.

```
\cs_set_nopar:Npn \luamplibnotagtextboxset #1 #2
{
  \bool_set_eq:NN \l_tmpa_bool \l__luamplib_tag_usetext_bool
  \bool_set_false:N \l__luamplib_tag_usetext_bool
  \tag_socket_use:nnn{luamplib/textext/set}{#1}{#2}
  \cs_gset_nopar:cpn {luamplib.notaggedbox.#1}{#1}
  \bool_set_eq:NN \l__luamplib_tag_usetext_bool \l_tmpa_bool
}
\sys_if_output_pdf:TF
{
  \cs_set_nopar:Npn \mplibputtextbox #1 #2 #3 #4
  {
    \pdfextension save\relax
    \vbox to 0pt{\vss
      \hbox bdir0 to 0pt{\kern #2bp \pdfextension setmatrix {#4}
        \socket_use:nnn{tagsupport/luamplib/textext/put}{#1}{\raise\dp#1\copy#1}\hss}
      \kern #3bp}
    \pdfextension restore\relax
  }
}
{
  \cs_set_nopar:Npn \mplibputtextbox #1 #2 #3 #4
  {
    \special{pdf:btrans~matrix~#4~#2~#3}
    \vbox to 0pt{\vss\hbox bdir0 to 0pt{
      \socket_use:nnn{tagsupport/luamplib/textext/put}{#1}{\raise\dp#1\copy#1}\hss}}
    \special{pdf:etrans}
  }
}
```

TODO: Not sure whether asgroup/mplibgroup with text mode will be tagged correctly. Prob-

ably not. At least, this will raise a warning.

```
\cs_set_nopar:Npn \luamplibtagasgroupset
{
  \bool_set_false:N \l__luamplib_tag_usetext_bool
}
\cs_set_nopar:Npn \luamplibtagasgroupput
{
  \bool_if:NT \l__luamplib_tag_usetext_bool { \tag_resume:n{\luamplibtagasgroupput} }
  \tag_socket_use:nnn{luamplib/mplibgroup/put}
}
```

A socket for mplibgroup. Again, we issue a warning upon text mode.

```
\socket_new:nn{tagsupport/luamplib/mplibgroup/put}{2}
\socket_new_plug:nnn{tagsupport/luamplib/mplibgroup/put}{default}
{
  \cs_if_free:cT {luamplib.mplibgroup.text.#1}
  {
    \msg_warning:nnn {luamplib} {mplibgroup-text-mode} {#1}
    \cs_gset_nopar:cpn {luamplib.mplibgroup.text.#1} {#1}
  }
  \tag_mc_end:
  \tag_mc_begin:n{tag=text}
  #2
  \tag_mc_end:
  \tag_mc_begin:n{artifact}
}
\socket_assign_plug:nn{tagsupport/luamplib/mplibgroup/put}{default}
```

A macro for BBox attribute

```
\cs_set_nopar:Npn \__luamplib_tag_bbox_attribute:n #1
{
  \tl_set:Ne \l_tmpa_tl {luamplib.BBox.\tag_get:n{struct_num}}
  \tex_savepos:D
  \property_record:ee{\l_tmpa_tl}{xpos,ypos}
  \tl_set:Ne \l__luamplib_BBox_llx_tl
    { \dim_to_decimal_in_bp:n { \property_ref:een {\l_tmpa_tl}{xpos}{0}sp } }
  \tl_set:Ne \l__luamplib_BBox_lly_tl
    { \dim_to_decimal_in_bp:n { \property_ref:een {\l_tmpa_tl}{ypos}{0}sp - \dp#1 } }
  \tl_set:Ne \l__luamplib_BBox_urx_tl
    { \dim_to_decimal_in_bp:n { \l__luamplib_BBox_llx_tl bp + \wd#1 } }
  \tl_set:Ne \l__luamplib_BBox_ury_tl
    { \dim_to_decimal_in_bp:n { \l__luamplib_BBox_lly_tl bp + \ht#1 + \dp#1 } }
  \bool_if:NT \l__luamplib_tag_bboxcorr_bool
  {
    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn
    {
      \l__luamplib_BBox_llx_tl
      \l__luamplib_BBox_lly_tl
      \l__luamplib_BBox_urx_tl
```

```
      \l__luamplib_BBox_ury_tl
    }
    {
      \int_incr:N \l_tmpa_int
      \tl_set:Ne ##1
      {
        \fp_eval:n
        {
          ##1
          +
          \dim_to_decimal_in_bp:n { \seq_item:NV \l__luamplib_tag_bboxcorr_seq \l_tmpa_int }
        }
      }
    }
  }
  \tag_struct_gput:ene {\tag_get:n{struct_num}} {attribute}
  {
    /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__tag_graphic_debug_bool
  {
    \iow_log:e
    {
      luamplib/tagging~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
    }
    \sys_if_output_pdf:TF
    {
      \tl_set:Ne \l__luamplib_tag_bbox_draw_tl
      {
        \pdfextension save\relax
        \opacity_select:n{0.5} \color_select:n{red}
        \pdfextension literal~text
        {
          \l__luamplib_BBox_llx_tl\c_space_tl
          \l__luamplib_BBox_lly_tl\c_space_tl
          \fp_eval:n { \l__luamplib_BBox_urx_tl - \l__luamplib_BBox_llx_tl }~
          \fp_eval:n { \l__luamplib_BBox_ury_tl - \l__luamplib_BBox_lly_tl }~
          re~f
        }
        \pdfextension restore\relax
```

```
      }
    }
    {
      \tl_set:Ne \l__luamplib_tag_bbox_draw_tl
      {
        \special{pdf:bcontent}
        \opacity_select:n{0.5} \color_select:n{red}
        \special{pdf:code~
          1~0~0~1~
          -\dim_to_decimal_in_bp:n { \property_ref:een{\l_tmpa_tl}{xpos}{0}sp + \wd#1 }~
          -\dim_to_decimal_in_bp:n { \property_ref:een{\l_tmpa_tl}{ypos}{0}sp }~
          cm
        }
        \special{pdf:code~
          \l__luamplib_BBox_llx_tl\c_space_tl
          \l__luamplib_BBox_lly_tl\c_space_tl
          \fp_eval:n { \l__luamplib_BBox_urx_tl - \l__luamplib_BBox_llx_tl }~
          \fp_eval:n { \l__luamplib_BBox_ury_tl - \l__luamplib_BBox_lly_tl }~
          re~f
        }
        \special{pdf:econtent}
      }
    }
  }
}
```

Sockets for main process

```
\socket_new:nn{tagsupport/luamplib/figure/begin}{1}
\socket_new:nn{tagsupport/luamplib/figure/end}{2}
\socket_new_plug:nnn{tagsupport/luamplib/figure/end}{transparent}{#2}
\socket_new_plug:nnn{tagsupport/luamplib/figure/begin}{alt}
{
    \tag_mc_end_push:
    \tl_if_empty:NT\l__luamplib_tag_alt_tl
    {
      \tl_if_empty:eTF{#1}
        { \tl_set:Nn \l__luamplib_tag_alt_tl {metapost~figure} }
        { \tl_set:Ne \l__luamplib_tag_alt_tl {metapost~figure~\text_purify:n{#1}} }
      \msg_warning:nnVV{luamplib}{alt-text-missing}
                       \l__luamplib_tag_envname_tl \l__luamplib_tag_alt_tl
    }
    \tag_struct_begin:n
    {
      tag=\l__luamplib_tag_struct_tl,
      alt=\l__luamplib_tag_alt_tl,
    }
    \tag_mc_begin:n{}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/end}{alt}
```

```
{
    \__luamplib_tag_bbox_attribute:n {#1}
    #2
    \tl_use:N \l__luamplib_tag_bbox_draw_tl
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin_pop:n{}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/begin}{actualtext}
{
    \tag_mc_end_push:
    \tag_struct_begin:n
    {
      tag=Span,
      actualtext=\l__luamplib_tag_actual_tl,
    }
    \tag_mc_begin:n{}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/end}{actualtext}
{
    #2
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin_pop:n{}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/begin}{artifact}
{
    \tag_mc_end_push:
    \tag_mc_begin:n{artifact}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/end}{artifact}
{
    #2
    \tag_mc_end:
    \tag_mc_begin_pop:n{}
}
```

A socket for tagging init, so that we can declare `\SetKeys[luamplib/tagging]{...}` anywhere in the document.

```
\socket_new:nn{tagsupport/luamplib/figure/init}{0}
\socket_new_plug:nnn{tagsupport/luamplib/figure/init}{alt}
{
  \socket_assign_plug:nn{tagsupport/luamplib/figure/begin}{alt}
  \socket_assign_plug:nn{tagsupport/luamplib/figure/end}{alt}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/init}{actualtext}
{
  \socket_assign_plug:nn{tagsupport/luamplib/figure/begin}{actualtext}
  \socket_assign_plug:nn{tagsupport/luamplib/figure/end}{actualtext}
```

In vmode, hmode will be forced by \noindent upon actualtext and text modes.

```
  \prependtomplibbox \mplibnoforcehmode
  \mode_if_vertical:T { \noindent \aftergroup\par }
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/init}{artifact}
{
  \socket_assign_plug:nn{tagsupport/luamplib/figure/begin}{artifact}
  \socket_assign_plug:nn{tagsupport/luamplib/figure/end}{artifact}
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/init}{text}
{
  \bool_set_true:N \l__luamplib_tag_usetext_bool
  \socket_assign_plug:nn{tagsupport/luamplib/figure/begin}{artifact}
  \socket_assign_plug:nn{tagsupport/luamplib/figure/end}{artifact}
  \prependtomplibbox \mplibnoforcehmode
  \mode_if_vertical:T { \noindent \aftergroup\par }
}
\socket_new_plug:nnn{tagsupport/luamplib/figure/init}{off}
{
  \socket_assign_plug:nn{tagsupport/luamplib/figure/begin}{noop}
  \socket_assign_plug:nn{tagsupport/luamplib/figure/end}{transparent}
}
\socket_assign_plug:nn{tagsupport/luamplib/figure/init}{alt}
```

Key-value options

```
\keys_define:nn{luamplib/tagging}
{
  ,alt .code:n =
  {
    \tl_set:Ne\l__luamplib_tag_alt_tl{\text_purify:n{#1}}
    \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{alt}
  }
  ,actualtext .code:n =
  {
    \tl_set:Ne\l__luamplib_tag_actual_tl{\text_purify:n{#1}}
    \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{actualtext}
  }
  ,artifact .code:n = { \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{artifact} }
  ,text     .code:n = { \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{text} }
  ,off      .code:n = { \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{off} }
  ,tag      .code:n =
  {
    \str_case:nnF {#1}
    {
      {false}    { \keys_set:nn {luamplib/tagging} {off} }
      {artifact} { \keys_set:nn {luamplib/tagging} {artifact} }
    }
    {
      \tl_set:Nn\l__luamplib_tag_struct_tl{#1}
```

```
      \socket_assign_plug:nn{tagsupport/luamplib/figure/init}{alt}
    }
  }
  ,adjust-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}
  }
  ,tagging-setup .code:n = { \keys_set_known:nn {luamplib/tagging} {#1} }
}
\keys_define:nn {luamplib/instance}
{
  ,instance     .code:n = { \tl_gset:Nn \currentmpinstancename {#1} }
  ,instancename .meta:n = { instance = {#1} }
  ,unknown      .code:n = { \tl_gset:NV \currentmpinstancename \l_keys_key_str }
}
```

Redefine our macros

```
\cs_set_nopar:Npn \mplibstarttoPDF #1 #2 #3 #4
{
  \prependtomplibbox
  \hbox dir~TLT\bgroup
    \tag_socket_use:nn{luamplib/figure/begin}\l__luamplib_tag_alt_dflt_tl
    \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
      \tag_suspend:n{\mplibstarttoPDF}
      \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
    \tag_socket_use:nnn{luamplib/figure/end}{\mplibscratchbox}{\box\mplibscratchbox}
  \egroup
}
\RenewDocumentCommand\mplibcode{O{}}
{
  \tl_set:Nn \l__luamplib_tag_envname_tl {mplibcode}
  \tl_gclear:N \currentmpinstancename
  \keys_set_known:neN {luamplib/tagging} {#1} \l_tmpa_tl
  \keys_set:nV {luamplib/instance} \l_tmpa_tl
  \tl_set_eq:NN \l__luamplib_tag_alt_dflt_tl \currentmpinstancename
  \tag_socket_use:n{luamplib/figure/init}
  \mplibtmptoks{}\ltxdomplibcode
}
\RenewDocumentCommand\mpfig{s O{}}
{
  \begingroup
  \tl_set:Nn \l__luamplib_tag_envname_tl {mpfig}
  \keys_set_known:ne {luamplib/tagging} {#2}
  \tl_set_eq:NN \l__luamplib_tag_alt_dflt_tl \mpfiginstancename
  \tag_socket_use:n{luamplib/figure/init}
  \IfBooleanTF{#1} { \mplibprempfig * }
                   { \mplibmainmpfig }
}
\RenewDocumentCommand\usemplibgroup{O{} m}
{
  \begingroup
  \tl_set:Nn \l__luamplib_tag_envname_tl {usemplibgroup}
  \keys_set_known:ne {luamplib/tagging} {#1}
  \tag_socket_use:n{luamplib/figure/init}
  \prependtomplibbox\hbox dir~TLT\bgroup
    \tag_socket_use:nn{luamplib/figure/begin}{#2}
    \setbox\mplibscratchbox\hbox\bgroup
      \bool_if:NF \l__luamplib_tag_usetext_bool { \tag_suspend:n{\usemplibgroup} }
      \tag_socket_use:nnn{luamplib/mplibgroup/put}{#2}{\csname luamplib.group.#2\endcsname}
    \egroup
    \tag_socket_use:nnn{luamplib/figure/end}{\mplibscratchbox}{\unhbox\mplibscratchbox}
  \egroup
  \endgroup
}
```

Allow setting alt/actual text within METAPOST code. Of course we can use them in TEX code as well.

```
\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

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

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.