The gmp package[*]

Enrico Gregorio
enrico DOT gregorio AT univr DOT it

March 10, 2011

Introduction

ConTEXt is far more superior to LATEX in its integration of METAPOST. On the other hand, few people are going to do the big jump towards ConTEXt, especially if they have to use maths in their documents.

Nonetheless, sometimes a better integration between LATEX and METAPOST would be welcome. One great package is mfpic, but its philosophy is different: it defines a set of TEX macros which are afterwards interpreted as METAPOST. It suffers a bit from its METAFONT origin: the user has to pass the dimensions of the created figure in advance, while METAPOST is able to figure them out by itself.

A very recent package does similar things as gmp, Vafa Khalighi's mpgraphics, from which I learned to add the `runs` and `clean` keys. However also Vafa's package can't pass parameters to the METAPOST pictures at run time. I have to thank Vafa Khalighi because his mpgraphics package helped me to straighten some roughness from the preliminary version of this package. Another package is oriented to this purpose, emp by Thorsten Ohl, but it has not been updated since 1997.

What I wanted was a package which could do *automatically* the compilation of METAPOST output and, moreover, could pass parameters to METAPOST. Wouldn't it be nice being able to define a graphic object in terms of the current text width? Or of the current baseline skip? Or of some user defined parameter? A "reusable" object which can change depending on current conditions? This package is an answer; it runs under LATEX, PDFLATEX and XƎLATEX.[1]

This has some consequences; for example, METAPOST labels will usually go inside

```
\btex ... etex
```

---

[*] This document corresponds to gmp v1.0, dated 2011/03/10.

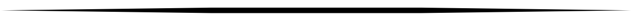
[1] It runs even under LuaLATEX, I'd say, but in this case it doesn't exploit the possibility of a direct call to the METAPOST library.

Figure 1 The code for the swelled rule

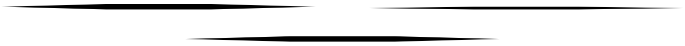
```
\begin{mpost}[name=swelled]
breadth=.667\mpdim{\linewidth};
height=2pt;
x1=0;
x2=x6=.333x4;x5=x3=.667x4;
x4=breadth;
y1=y4=height/2;
y2=y3=height;
y5=y6=0;
fill z1--z2--z3--z4--
  z5--z6--cycle;
\end{mpost}
```

instead of the normal `btex ... etex`. Later examples will show this is an enrichment, rather than a limitation.

Consider a swelled rule: we want its width to be two thirds of the line width. But, of course, we should not simply scale it when the line width changes. Here is an example; the swelled rule for the current line width:



We follow it with a scaled version corresponding to half line width (on the left) and a true swelled rule constructed for a halved line width:



The difference is clear, but I must admit to have cheated: in this case it would be sufficient to scale the swelled rule only in the horizontal direction. However, scaling is not always the correct answer, since it affects all rule widths and character sizes, for example. You can see the input for the swelled rule in Figure 1, taken from Peter [3].

The command `\mpdim` takes as an argument a length (rigid or rubber) and transforms it into a dimension understood by METAPOST (i.e., it strips off possible 'plus' and 'minus' specifications). To METAPOST eyes, it is an 'unnamed variable' like '`whatever`', except that it has the value of the dimension given as argument; see the description in Section 5

What I did was to define a METAPOST input for the swelled rule, to assign it a name and to use it when needed. For example, the above verbatim environment surrounded by swelled rules has been written as in Figure 2.

A recent message on `comp.text.tex`[2] asked how to realize in LaTeX chapter

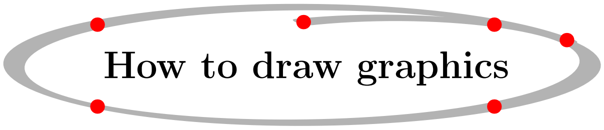
---

[2] `http://groups.google.com/group/comp.text.tex/browse_frm/thread/516fb024bb83bcda`

Figure 2 The input for the nice verbatim

```
\begin{center}
\usempost{swelled}\\*[3ex]
\begin{minipage}{.5\linewidth}
\begin{verbatim}
\begin{mpost}[name=swelled]
breadth=.667\mpdim{\linewidth};
...
\end{mpost}
\end{verbatim}
\end{minipage}\\*[3ex]
\usempost{swelled}
\end{center}
```

headers like the ones in the manual "ConTEXt, an excursion"[3] and Taco Hoekwater kindly provided the code that's used. Here I propose a macro that solves the problem, once it's integrated in a chapter style for memoir or titlesec; the METAPOST code is found in table 1 and the result is shown here



Now you can return to the first page and see that the title for the introduction was typeset inside a similar drawing, but of different size. The METAPOST code is exactly the same as Taco's, except for the call of `mpost` instead of `\startMPgraphic` and `\endMPgraphic`. The LaTeX code used is

```
\newsavebox{\tacochapterbox}
\newcommand{\tacochapterhead}[1]{%
  \sbox\tacochapterbox{\Large\bfseries #1}%
  {\ooalign{%
     \MPclipOne{\mpdim{\wd\tacochapterbox+6pc}} % width
               {\mpdim{\ht\tacochapterbox+3pc}} % height
               {8}                              % thickness of the curve
               {(.7,.7,.7)}                     % color of the curve
               {red}                            % color of the spots
     \cr\hfill\raisebox{\dimexpr.5\ht\tacochapterbox+1.5pc\relax}
          {\box\tacochapterbox}\hfill}}}

\tacochapterhead{How to draw graphics}
```

Use a high level package such as titlesec in order to draw fancy ornaments around the chapter title; an example might be

---

[3] `http://www.pragma-ade.nl/general/manuals/mp-cb-en.pdf`

```
\newcommand{\chapformat}[1]{\tacochapterhead{\thechapter.\ #1}}
\titleformat{name=\chapter}[block]
  {\large\bfseries\filcenter}{}{0pt}{\chapformat}
\titleformat{name=\chapter,numberless}[block]
  {\large\bfseries\filcenter}{}{0pt}{\tacochapterhead}
```

which takes care also of numberless chapter headings.

Another useless application comes from the Metafun manual, by Hans Hagen [1]: a representation of a counter using vertical bars with a diagonal one after four of them, just like when counting on a piece of scratch paper. We can borrow the METAPOST input from Hans, and define the command you find in table 2.

Now we have a new environment and we can use it at every nesting level:

1. A first item, to remember that METAPOST is a big fun, for the following reasons:
    - / it has a syntax very similar to METAFONT;
    - // it outputs very good PostScript;
    - /// there are many people using it;
    - //// it is very simple to use;
    - 𝍸 it integrates easily with LaTeX;
    - 𝍸 / some other reason just to arrive to six.
2. A second item, to remember that LaTeX users can benefit from the use of METAPOST.

The input for this nested enumeration was

```
\begin{enumerate}
  \item A first ...
    \begin{scratchenum}
      \item it has ...
      ...
    \end{scratchenum}
  \item A second ...
\end{enumerate}
```

Of course, if we need to use the `scratchenum` only at the first level of nesting, a definition such as

```
\newenvironment{scratchenum}
  {\begin{enumerate}
   \renewcommand{labelenumi}{\scratchcount{\arabic{enumi}}}%
  }
  {\end{enumerate}}
```

would have sufficed. We give as argument to `\scratchcount` the representation of the counter as an arabic number; METAPOST does all the necessary calculations.

Table 1 METAPOST code for a fancy chapter heading

```
%macro arguments:
%#1 = desired width
%#2 = desired height
%#3 = pen thickness (relative)
%#4 = line color
%#5 = dot color
\def\MPclipOne#1#2#3#4#5%
   {\begin{mpost}
      w := #1;  width  := 100;  wfactor := w/width;
      h := #2;  height := 100;  hfactor := h/height;
      color lightred;  lightred  := (.90,.50,.50);
      color lightgray; lightgray := (.95,.95,.95);
      color gray;      gray      := (.50,.50,.50);
      def random_delta (expr d) =
        d-(uniformdeviate 2d)
      enddef;
      z1 = (0,height);
      z2 = (0,0);
      z3 = (width,0);
      z4 = (width,height);
      z5 = (width+random_delta(.2width),height+random_delta(.2height));
      z6 = (.5width+random_delta(.1width),height+random_delta(.1height));
      pickup pencircle
        xscaled (#3/wfactor)
        yscaled (#3/(2*hfactor))
        rotated 30;
      draw z5..z1..z2..z3..z4..z6 withcolor #4;
      pickup pencircle
        xscaled (#3/wfactor)
        yscaled (#3/hfactor);
      draw z1 withcolor #5;
      draw z2 withcolor #5;
      draw z3 withcolor #5;
      draw z4 withcolor #5;
      draw z5 withcolor #5;
      draw z6 withcolor #5;
      newwidth  := (xpart (urcorner currentpicture)) -
                   (xpart (llcorner currentpicture));
      newheight := (ypart (urcorner currentpicture)) -
                   (ypart (llcorner currentpicture));
      currentpicture := currentpicture
       xscaled (w/newwidth) yscaled (h/newheight);
    \end{mpost}}
```

Table 2 The `\scratchcount` macro

```
\newcommand{\scratchcount}[1]{%
  \begin{mpost}
  n:=#1;
  height := 3/5\mpdim{\baselineskip} ;
  span := 1/3 * height ;
  drift := 1/10 * height ;
  pickup pencircle scaled (1/12 * height) ;
  def d = (uniformdeviate drift) enddef ;
  for i := 1 upto n :
    draw
      if (i mod 5)=0 : ((-d-4.5span,d)--(+d-0.5span,height-d))
      else : ((-d,+d)--(+d,height-d)) fi
      shifted (span*i,d-drift) ;
  endfor;
  picture cp ;
  cp := currentpicture ; %for readability
  setbounds currentpicture to
    (llcorner cp shifted (0,-ypart llcorner cp) --
     lrcorner cp shifted (0,-ypart lrcorner cp) --
     urcorner cp -- ulcorner cp -- cycle) ;
  \end{mpost}}
\makeatletter
\newenvironment{scratchenum}{\begin{enumerate}
  \@namedef{label\@enumctr}{\scratchcount{\arabic{\@enumctr}}}%
  }{\end{enumerate}}
\makeatother
```

Figure 3 Expanded definition for the scratch numbers; the `\VCEN` macro is for centering the object.

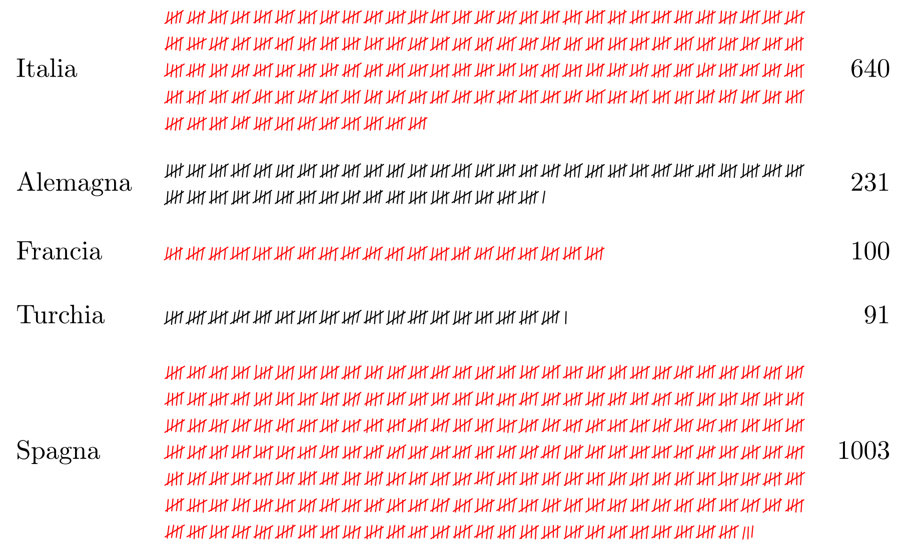
```
\newcommand\lista[2][black]{%
\VCEN{\begin{mpost}[mpmem=metafun]
n:=#2 ; color fuzzy_color ; fuzzy_color:=#1 ;

height := 5pt ;
span := 1/3 * height ;
drift := 1/10 * height ;
hsize := .7\mpdim{\linewidth} ;
vstep := 10pt ;
xmax := hsize div 5span ;

pickup pencircle scaled (1/12 * height) ;
def d = (uniformdeviate drift) enddef ;
for i := 1 upto n :
  xpos := ((i-1) mod (5*xmax))*span ;
  ypos := ((i-1) div (5*xmax))*vstep ;
  draw
    if (i mod 5)=0 : ((-d-4.5span,d)--(+d-0.5span,height-d))
    else : ((-d,+d)--(+d,height-d)) fi
    shifted (xpos,-ypos+d-drift) withcolor fuzzy_color ;
endfor;

picture cp ;
cp := currentpicture ; % for readability
if (ypart ulcorner cp - ypart llcorner cp) <= vstep :
setbounds currentpicture to
  (llcorner cp shifted (0,-ypart llcorner cp) --
   lrcorner cp shifted (0,-ypart lrcorner cp) --
   urcorner cp -- ulcorner cp -- cycle) ;
fi
\end{mpost}}}
```

Figure 4 La lista



We can expand on this theme, to represent greater numbers. You can see it in Figure 3. After this, an input like

```
\begin{tabular}{llr}
\multicolumn{2}{c}{\textbf{La lista}}\\\hline\\
Italia & \lista[red]{640} & 640\\\\
Alemagna & \lista{231} & 231\\\\
Francia & \lista[red]{100}& 100\\\\
Turchia & \lista{91}& 91\\\\
Spagna & \lista[red]{1003} & 1003\\\\\hline
\end{tabular}
```

gives what you can see in figure 4, but if we put it inside a `minipage`, we can render the same list in other ways, automatically. You see it in Figure 5.

Note how we can pass to the METAPOST input some parameters, a dimension as before, and a color specification.

It is not necessary to put `mpost` environments inside a definition, as we have already seen. We can assign a name to the object and use it as many times as we like, even subject to transformations with the same syntax as `\includegraphics`. The following example is from the METAPOST package drv, which requires a double run of METAPOST:

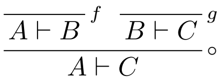
```
\begin{mpost}[runs=2,mpsettings={input drv;},name=der1]
jgm 0 "A\vdash B";
jgm 1 "B\vdash C";
jgm 2 "A\vdash C";
nfr 0 () ("f", 1);
```

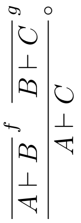
Figure 5 The tables have been reduced at 70%. On the right the table has been put in a `minipage` of size `.5\linewidth`; on the left the size was `.4\linewidth`

	La lista
IT	𝍸 𝍸
DE	𝍸 𝍷
FR	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸
TR	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍷
ES	𝍸 𝍷𝍷𝍷

	La lista
IT	𝍸 𝍸
DE	𝍸 𝍷
FR	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸
TR	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍷
ES	𝍸 𝍷𝍷𝍷

```
nfr 1 () ("g", 1);
nfr 2 (0, 1) ("\circ", 1);
draw drv_tree;
\end{mpost}
\usempost{der1}\quad
\usempost[angle=90]{der1}
```





If a METAPOST object is not assigned a name it is immediately used and, obviously, it cannot be used any more. Well, this is not strictly true: every object has a number corresponding to an MPS file named ⟨*filename*⟩`+mp`⟨*number*⟩`.mps` and it would be possible to include it in the usual way. Here ⟨*filename*⟩ is the name of the root LaTeX file. However, the ⟨*number*⟩ is not fixed, since insertion of other objects before a certain one will change its assigned number.

There are three ways for typesetting a document using `gmp`, which correspond to the following package options:

`shellescape` exploits the 'shell escape' feature present in most TeX distributions: every `mpost` environment calls a run of METAPOST, setting the object immediately (you have to call the typesetting engine with the necessary command line option, see later);

`noshellescape` defers the METAPOST runs to the end of the LaTeX run, the user has to launch them by hand—in this case a very simple shell script is written for doing all the business;

`nowrite` inhibits writing METAPOST files and disables running METAPOST: when the document is finished and no more modifications to the METAPOST objects are needed, we can set this option and read the MPS files already compiled in previous runs.

The 'shell escape' feature is very handy, but someone may feel uneasy with it. For security reasons, TeX distributions usually disable it by default and it must be explicitly requested by calling

```
latex -shell-escape ⟨filename⟩
pdflatex -shell-escape ⟨filename⟩
xelatex -shell-escape ⟨filename⟩
```

depending on what engine you are using; see your system's documentation for the actual call: for TeX Live the command line option `-shell-escape`, while for MiKTeX it's `-enable-write18`.

In case some METAPOST graphics doesn't compile properly, it's better to switch to `noshellescape` and launch the shell script after a new LaTeX run: from a terminal window say `sh` ⟨*filename*⟩`+mp.sh`, where ⟨*filename*⟩ stands for your main document's name. This will run METAPOST in 'error stop mode', so that you'll probably get a clue about what is going wrong.

If one uses XeLaTeX, all MPS files will be converted to PDF via `epstopdf`.

1 The name of the package and its workings

I wanted a successor of emp, but fmp was out of the question: this is a package for including functional METAPOST in LaTeX. So I chose gmp.

Here is a brief description of how the package works. A METAPOST object is defined through a `mpost` environment (see Section 7). When such an environment is found, an external METAPOST file is written and processed (either immediately or after the LaTeX run, at the user's choice). After processing, a file called ⟨*filename*⟩`+mp`⟨*number*⟩`.mps` is produced, which is then included in the LaTeX output with `\includegraphics`, but without user's intervention. If the object is assigned a name, it can be included as many times as desired.

The gmp package depends on xkeyval, ifxetex, ifpdf and, of course, graphicx. If you have to pass options to this one, load it before gmp.

2 File names

This package *doesn't* support file names containing spaces. Quoting D. E. Knuth, if you're upset about this, you shouldn't be: spaces in file names might be supported, but every system has its idiosyncrasies and quotes around the name aren't always the correct answer. Avoiding strange characters in file names is the best strategy for being compatible with most platforms.

3 To do

Find out a way to coerce creation of all auxiliary files in a subdirectory of the current directory, in order not to fill it with junk, besides what's really necessary.

4 The package options

We have seen some of the package options, but now I will present them more formally. Some of them will be grouped together; the first one of the group is the default.

`tex` and `latex`

METAPOST is able to use either Plain TeX or LaTeX for typesetting the 'labels'. If you are using ten point type and the usual Computer Modern fonts or no label at all, it is not necessary to set this option (the default is `tex`). On the other hand, if you are using different fonts or different type sizes, then `latex` is necessary. The package creates a crude preamble keeping the type size of your document; insertion of any other package or of a different class must be explicitly done with the commands `\usempxclass` and `\usempxpackage` which have the same syntax as `\documentclass` and `\usepackage`. See Section 5 for a description of the commands; their names should remember that they are used for an 'extension' of METAPOST, namely the preprocessor that typesets the labels making them into METAPOST pictures. However the option can be overridden for any picture, see later on.

`noshellescape` and `shellescape`

The `shellescape` option uses the `\write18` feature found in many TeX distributions (all the up-to-date ones based on Web2C such as TeX Live, and MiKTeX). Its purpose is to stop momentarily the TeX compilation for executing the METAPOST run after a `mpost` environment has been found. LaTeX or PDFLaTeX have to be called with the `-shell-escape` command line option or `-enable-write18` on MiKTeX.

With `noshellescape` all shell commands are collected in a batch file and LaTeX will remember users to execute it at the end of the run. On the subsequent LaTeX run, all objects should be in place.

When a compiled METAPOST object is not found at the proper place, a box like the one in the margin is printed. The same will happen in presence of compilation errors, precisely when the proper MPS file is not found.

MP

When using the `noshellescape` option, it may happen that after a compilation the objects are misplaced, for example if some new object has been added; running METAPOST on the created files and another LaTeX run should fix this. The default option is `noshellescape`.

`write` and `nowrite`

The `nowrite` option is useful when all METAPOST objects are present in the document, they are in their final form and all of them have been processed. By setting the `nowrite` option, no writing of METAPOST files and no compilation take place, speeding up the process. I would have called this option `final` and its contrary `draft`, but this usage conflicts with the workings of graphicx. The default option is `write`, so it's not necessary to specify it.

`envname`

You can change the name of the environment for the METAPOST objects; the default is `mpost`. If you don't like it or you want this name for your favorite environment, you can say

`envname=`⟨*string*⟩

and ⟨*string*⟩ will be used for introducing METAPOST objects. This key must be specified only as a package option, not in the argument of `\gmpoptions`.[4]

`extension`

If the root LaTeX file is called `file`, then the compiled MPS files will be named `file+mp`⟨*number*⟩`.mps`. The string `+mp` is the *mp-extension*, which can be modified by saying

`extension=`⟨*string*⟩

in the options to `gmp`; the ⟨*string*⟩ should consist only of printable ASCII characters with no special meaning to TeX. So, with a main file called `paper.tex`, the created files will be named `paper+mp0001.mp` and so on; with

```
\gmpoptions{extension=--mp--}
```

the names will be `paper-mp-0001.mp` and so on. *Don't use a slash* in the mp-extension! Leave this option to the default value, unless you happen to have files whose name contains the string `+mp`.

`everymp`

Here you can set whatever you want to be present in all METAPOST objects, for example input files. Any set of METAPOST statements can be given in braces, for example

```
everymp={input boxes;}
```

Since these are METAPOST statements, don't forget the trailing semicolon. If you have a long list of METAPOST macros that should be loaded for each picture, it's better to save them in a file and input this one with the `everymp` key. When your document uses font packages, you may need to say

```
everymp={prologues=3;}
```

so that your MPS files will contain correct references to the fonts; this is automatically done when the typesetting engine is XeLaTeX.

`clean=none|aux|mp`

After the compilation, there will be many auxiliary files around. The essential ones are only those necessary for the picture inclusion, namely the MPS or PDF final results. With `clean=none` nothing will be removed, with `clean=aux` the files with extension `.log`, `.mpx` and `.mpo` created during the METAPOST runs will be deleted. Also the `.mp` files will be deleted with `clean=mp`; this option will erase also the `.mps` files when XeLaTeX is used.

---

[4] Actually it can be in `\gmpoptions`, but there it has no effect.

Some strange files will remain, particularly when there's some error during one of the METAPOST runs; this is why I chose not to delete them. It's best to use `clean=none`, which is the default, at the beginning, changing to `clean=aux` when everything seems to be alright and `clean=mp` for the last check before setting `nowrite`.

In case you are worried about what files will be deleted, run with the `noshellescape` options and have a look at the shell script that's created: it will show at its end the remove commands that will be run. Or just don't set anything, so that `clean=none` will be in force.

`rmcommand`

The default command for cleaning the directory is `rm -f`. Of course this is system dependent and those on Windows™ systems might want to set `rmcommand=del`.

`postrmcommand`

The value of the key `postrmcommand` is usually empty; you can obtain a 'safe removing' by setting something like

```
\gmpoptions{rmcommand=mv,postrmcommand=junk}
```

This will be translated into commands such as

```
mv filename+mp*.log junk
```

so if you have a subdirectory called `junk` all the auxiliary files will be moved into it, instead of being deleted.

5 Commands

`\gmpoptions{`⟨*options*⟩`}`

All package options can be given either as an optional argument to `\usepackage` or as an argument to `\gmpoptions`. This command can be given multiple times, with cumulative effect, but only in the preamble. However the `envname` option has no effect here.

`\usempxclass [`⟨*options*⟩`] {`⟨*class*⟩`}`

This has the same syntax as `\documentclass`. Its purpose is to declare the class used for the runs of TEX or LATEX for preparing the `mpx` files which METAPOST needs to draw the text labels. It should be not necessary to change it from the default (which is `article`). But if you are using a non standard class with a non standard type size (i.e., not 10, 11 or 12 point size), then you *will need* to declare a class, otherwise your labels will be typeset in 10 point type. Of course this has no effect for METAPOST objects where the TEX format to be used is Plain TEX.

`\usempxpackage [`⟨*options*⟩`] {`⟨*package*(*s*)⟩`}`

Same syntax as `\usepackage`. It can be given multiple times and its effects are cumulative. Thus, if your main text font is Palatino, you have to give

```
\usepackage{palatino}
\usempxpackage{palatino}
```

I chose not to modify the standard `\usepackage` command because some packages make no sense inside METAPOST objects and it would be useless and time consuming to load them. Just think to the `pgf` package, for example: while a user may very well employ this one alongside with `gmp`, it would be very questionable to load `pgf` during a LaTeX run to compile a METAPOST graphic object, albeit not forbidden.

If your labels in METAPOST pictures use special characters (the accented ones, for instance), you *must* load the `inputenc` package *with the same option* as your main file:

```
...
\usepackage[latin1]{inputenc}
...
\usepackage{gmp}
\gmpoptions{...}
\usempxpackage[latin1]{inputenc}
...
```

You can use any encoding with (PDF)LaTeX; set the `utf8` or `utf8x` option when your main document is typeset with XeLaTeX.

Both `\usempxpackage` and `\usempxclass` respect the usual scoping rules, so you can modify the class or the packages used for a particular METAPOST object by enclosing the commands in a group, for example in an environment such as `figure`. But no, not in the `mpost` environment itself, sorry.

`\resetmpxpackages`

This command resets to empty the list of packages to be loaded, obeying to the normal scoping rules.

`\mpxcommands{`⟨*commands*⟩`}`

With this command you can add LaTeX commands to be executed for every run of METAPOST, when typesetting labels; for instance, your favorite definitions. Also the effect of this command is cumulative. Pay attention that this works only if the program used for compiling the `mpost` figures is LaTeX. Like the commands for choosing the class and the packages, this one obeys the scoping rules.

`\resetmpxcommands`

This command resets the list of commands to empty, obeying to the normal scoping rules.

`\mpdim{`⟨*dimen parameter*⟩`}`

This command takes as argument a length, i.e., a LaTeX parameter or a command defined by `\newlength` or operations on lengths. Its output is the length in points; if the length is a rubber one, the 'plus' and 'minus' parts are ignored. You can even use as argument an explicit length such as '`2cm`', although this would be overkill. However, you can give as argument some calculation with `\dimexpr` syntax, such as

```
\mpdim{2ex+\baselineskip}
\mpdim{3em-1pt}
```

which might be useful, because the `ex` and `em` units change with the current font size. Don't add `\relax`, it's automatically provided.

TeXnical note. The command `\mpdim` works without making assignments, so that it is fully expandable at the moment of the `\write` to the METAPOST external file. It uses internally `\the` and `\dimexpr`. When the `\write` takes place, the command is expanded and the output is

`begingroup` ⟨*thedimen*⟩ `endgroup`

where ⟨*thedimen*⟩ is the value of the ⟨*dimen*⟩ given as argument. Therefore the construction `\mpdim{`⟨*dimen*⟩`}` behaves like a number as far as METAPOST is concerned; for Metapost, dimensions are simply numbers and the units are just multipliers.

`\usempost`

This command takes as its argument a string already assigned as a name to a METAPOST object. Its effect is to place the graphic object where the command appears. The complete syntax is

`\usempost[`⟨*options*⟩`]{`⟨*string*⟩`}`

where ⟨*string*⟩ is the name of an already introduced and named METAPOST object and ⟨*options*⟩ are any options that make sense for the `\includegraphics` command provided by the graphicx package.

6 Environments

The package provides two environments, whose default names are `mpost` and `mpost*`. The name, however, can be changed in the package options, as we have already seen. In the descriptions below I'll use the default name.

Commands in the body of the `mpost` environment *are* interpreted and expanded and this is the main feature offered by this package.

The main difference between `mpost` and `mpost*` is that in the second one there is no expansion at all: every METAPOST statement in the environment's body is written out as it is for subsequent processing. Therefore `mpost*` behaves like `emp` or `mpgraphics` of the respective packages, but the difference is that it *can* go in the argument to a command. This behavior costs us something: LaTeX environments in METAPOST labels *must* be enclosed in a pair of braces in the same `btex...etex` construction. So don't say

```
btex ... $\begin{array}{c}a\\b\end{array}$ ... etex
```

but rather

```
btex ... {$\begin{array}{c}a\\b\end{array}$} ... etex
```

See later on for details. The same holds for the `mpost` environment, with one more little catch, see page 21.

There is no `\inputmpost` command: if you really need it, just define

```
\newcommand{\inputmpost}[2][1]{\begin{mpost}[#1]input #2;\end{mpost}}
```

The METAPOST file you input this way should contain just statements that would go in a normal `mpost` environment.

7 Options to the environments

Actually, the name of the two environments provided by this package can be chosen by the user on a per document basis with the package option `envname`. I will use the default name in the description. The syntax is

`\begin{mpost}[`⟨*options*⟩`]`
⟨*METAPOST statements*⟩
`\end{mpost}`

`\begin{mpost*}[`⟨*options*⟩`]`
⟨*METAPOST statements*⟩
`\end{mpost*}`

The ⟨*options*⟩ are in the form ⟨*key*⟩=⟨*value*⟩. They influence only the current environment and are the same for the two varieties (`mpost` and `mpost*`).

name

With `name=`⟨*string*⟩ you assign the object a name for subsequent multiple use with `\usempost{`⟨*string*⟩`}`. For example, if the object is some symbol to be employed many times, you can say

```
\begin{mpost}[name=mysymbol]
...
\end{mpost}
\newcommand{\mynicesymbol}{%
  \usempost[height=.7\ht\strutbox]{mysymbol}}
```

In this case you cannot pass arguments to the `mpost` environment: the object is *frozen* apart from modifications with the optional argument to `\usempost`. In this case the symbol we have built with METAPOST will change size according to the current font size, but only by scaling.

The ⟨*string*⟩ used as a name should consist only of ASCII printable characters; no accented letters, please, just like for LaTeX labels (those for `\label`). If you specify this key, then the object will not appear unless you specify also `use`, see below. The proper place for named environments is usually the preamble.

Suppose you have to write the symbol for a square wave in different sizes; instead of using only one MPS graphic file and scaling it you may want to define a couple of them. Say, for instance, that you need it also in section titles, which are typeset in `\Large` size and bold face and that the symbol's height is the same as a lowercase letter. You can get the two versions which are automatically chosen among with the following code:

```
\makeatletter
\newcommand{\sqwave}{\usempost{sqwave\f@size\f@series}}
\newcommand{\sqwavebody}{%
  u:=\mpdim{1ex};
  pickup pencircle scaled \mpdim{.05em};
  draw (0,0)--(2/3u,0)--(2/3u,u)--(u,u)--(u,0)--(5/3u,0);}
{\normalsize
\begin{mpost}[name=sqwave\f@size\f@series]\sqwavebody\end{mpost}
}
{\Large\bfseries
\begin{mpost}[name=sqwave\f@size\f@series]\sqwavebody\end{mpost}
}
\makeatother
```

Then simply say `\sqwave` when you need the symbol. If you change the type size of your document, you'll get the proper symbol's height. If it turns out that you need the symbol also in other contexts, just add other `mpost` named environments along the same lines.

We pass the `mpost` environment two dimensions that depends on the context: the symbol's height will be the x-height of the current font, the pen's width will scale according to the em size, which is bigger for bold face fonts. The picture will be chosen according to the current value of `\f@size` (the current type size) and `\f@series`, which contains the font series indicator ('m' for medium fonts and 'bx' for bold face fonts, usually). The auxiliary macro `\sqwavebody` contains the body of the `mpost` environment, so there's even no need to copy it many times.

use

This is a boolean option, the default is `true` unless the `name` option is present. When it is set to `true`, the object is inserted in the place where the environment appears. In order to set it to `true`, it is sufficient to say `use` instead of `use=true`.

mpmem

You can choose a different format for METAPOST compilation instead of the default Plain. For example you can use Hans Hagen's Metafun package by passing the option

```
mpmem=metafun
```

mpxprogram

With this option, you can override the global option chosen when `gmp` was loaded, or with `\gmpoptions`; with `mpxprogram=tex` you invoke Plain TEX, with `mpxprogram=latex` you invoke LATEX with the preamble defined by default or in the options to `gmp`. There are presently no other choices. Use `mpxprogram=latex` (or set the package option `latex`) whenever you have METAPOST labels.

mpxcommands

You can pass some commands to be executed when compiling METAPOST with LATEX with `mpxcommands=`⟨*commands*⟩. For instance, you can say

```
mpxcommands={\newcommand{\C}{\mathbf{C}}}
```

if your labels contain a `\C` command for typesetting the set of complex numbers in bold face type or some other useful commands. You may also say

```
mpxcommands={\input{mymacros}}
```

and put all needed definitions inside `mymacros.tex`. Remember that inside `\btex...etex` constructions (see Section 8), METAPOST does *not* know any of your personal commands, since the typesetting is made during a distinct run of LATEX. The LATEX commands are not interpreted during the write to the auxiliary file, they will be during METAPOST compilation.

mpsettings

The value for this key is a list of METAPOST statements to be executed *before* `beginfig`, usually for inputting METAPOST packages. For example, if a METAPOST object is drawn with the `boxes` package, then it is necessary to say

```
mpsettings={input boxes;}
```

because this package must be loaded before saying `beginfig`. Of course such a statement can be included also with the `everymp` global key. Don't forget the trailing semicolon. LATEX commands are interpreted just as in the `mpost` environment, so it's possible to use, for example, `\mpdim`.

runs

The `drv` package for METAPOST requires each source file to be run twice; in this case set the option `runs=2`. For this specific package it's necessary to use also the `latex` option and also, of course, `mpsettings=input drv;` in order to load the METAPOST package. The default value is 1.

8 The body of the `mpost` and `mpost*` environments

The body of the `mpost` and `mpost*` environments consists of METAPOST statements with the exception of `beginfig(`⟨*number*⟩`);`, `endfig;` and `end`[5].

The METAPOST input that will be processed consists of

⟨*TEX preamble*⟩
⟨*METAPOST preamble*⟩
`beginfig(`⟨*number*⟩`);`
⟨*body of the environment*⟩
`endfig;end`
⟨*TEX postamble*⟩

The ⟨*body of the environment*⟩ is precisely the contents of the `mpost` environment you supply. The ⟨*number*⟩ is assigned automatically and is actually irrelevant.

The ⟨*TEX preamble*⟩ is empty if the compiler for the `mpx` files is TEX; it is the LATEX preamble formed as seen before in case the compiler is LATEX. In the latter case, the ⟨*TEX postamble*⟩ is just

```
verbatimtex \end{document} etex
```

otherwise it is empty (the METAPOST interpreter knows to add `\end` at the end of its produced `mpx` file, if necessary). Of course the ⟨*TEX preamble*⟩ is complemented by the declarations you supply with the previously described commands. Similarly, the ⟨*METAPOST preamble*⟩ is usually almost empty, but you may add something with the methods described earlier; actually the only statement always added to the preamble is

```
outputtemplate:= "%j.mps";
```

in order to give the resulting file a suitable name for automatic inclusion in the LATEX document. Don't issue such a statement yourself.

In the ⟨*body of the environment*⟩ you can put TEX or LATEX commands as usual for METAPOST code inside `btex` ⟨*TEX material*⟩ `etex`; in a `mpost` environment, however, LATEX commands may be embedded in the normal METAPOST statements, we'll call them *naked*.

Naked TEX commands in the body of `mpost`

The *naked* commands will be expanded in the normal way, it's up to you to ensure that their expansion is valid METAPOST code. Don't use dimension parameters by themselves: for example, the command `\linewidth` is *unexpandable*, so METAPOST will interpret it as the variable `linewidth`, which probably would be unknown.

Suppose you want that your METAPOST graphic objects have a uniform aspect throughout the document, but you haven't decided yet what will be the final background color for a series of pictures. All you need is to use something like

```
fill p withrgbcolor \backgroundcolor;
```

---

[5] Actually, you *can* put such statements; but don't blame me if your computer starts choking or insulting you.

and to give in the document's preamble a definition such as

```
\newcommand*{\backgroundcolor}{(.124,.048,.129)}
```

and your backgrounds will be the same in all the pictures. Of course there are other methods to get this result: one might say also

```
\gmpoptions{everymp={rgbcolor mybgcolor;
   mybgcolor:=(.124,.048,.129)}}
```

or write a definition file to be read via `everymp`; but I believe that the *naked command* method is preferable from the point of view of a LaTeX user: it requires less maintenance and is more direct.

In order to pass length parameters to METAPOST, the `\mpdim` command is useful. If you know that the parameter is a fixed length (such as `\linewidth`), you can use also `\the` before the name of the length parameter. Beware however that most parameters in LaTeX are defined through `\newlength` and so they are *rubber* length, possibly having 'plus' or 'minus' components which would confuse METAPOST. Using `\mpdim` is safer. Moreover, `\mpdim` accepts its argument in the usual braced form.

Pay attention that the expansion of `\mpdim` is not only the length in points, but something more complicated. Don't use it outside `mpost` environments.

Any command whose expansion is just a string of characters can be used naked in a `mpost` environment; METAPOST will see the string of letters and it's under your responsibility that this string is meaningful in the context it's found. For example, after

```
\newcommand{\xxx}[1]{#1cm}
```

has been given in the LaTeX root file, `\xxx{2}` can be written in a `mpost` environment, where METAPOST expects a numeric value; for instance,

```
a:=\xxx{2};
```

would set the variable '`a`' to the value '`56.6929`', which are indeed two centimeters, since METAPOST works internally in 'big points'. Pay attention that

```
a:=2\xxx{2};
```

would set the variable '`a`' to the value '`623.62192`' which corresponds to 22 centimeters and not to '`113.3858`' (i.e., 4 centimeters). The correct way to set '`a`' to 4 centimeters would be `a:=2*\xxx{2};`.

Commands inside `[verbatim|btex]...etex`

This is an important point where the METAPOST input in a `mpost` environment deviates slightly from the normal one. Suppose that you have to write a label inside METAPOST in the form

```
btex \textit{label} etex
```

because you want it in italics. We have a problem here, because the command `\textit` passes through a `\write`, so that it is expanded to something that is not what the METAPOST interpreter likes to see; precisely, the expansion might be something like

```
\protect \relax \protect \edef n{it}\protect\xdef
\OT1/cmr/m/n/10 {\OT1/cmr/m/n/10 }\OT1/cmr/m/n/10
\size@update \enc@update
```

(all on one line). This will confuse the postprocessors that typeset the labels (that currently are integrated into METAPOST), either if we are using LaTeX or, even worse, Plain TeX.

The solution is to write `\verbatimtex` and `\btex` instead of `verbatimtex` and `btex`. The leading backslash would be legal also in standard METAPOST, anyway. Everything inside

```
\verbatimtex ... etex
\btex ... etex
```

is passed as is to the compiler for producing the `mpx` files needed by METAPOST. So the correct way to pass the label is

```
\btex \textit{label} etex
```

when post-processing with LaTeX and

```
\btex \it label etex
```

when post-processing with Plain TeX. Though using `btex` without a backslash has its uses, remember that eight bit characters *must* go inside `\btex...etex` or protected by `\unexpanded` (unless you're using XeLaTeX). So, never say

```
btex Cuvéé 1982 etex
```

but choose among the following alternatives:

```
\btex Cuvéé 1982 etex
btex \unexpanded{Cuvéé} 1982 etex
btex Cuv\unexpanded{éé} 1982 etex
```

TeXnical note. When I started developing this package, the choice to write verbatim the METAPOST input was ruled out almost immediately, since I wanted to pass parameters from LaTeX to METAPOST. The verbatim method is used by `emp`, which makes available only two parameters given as arguments to the `emp` environment; for example `\begin{emp}(23,45)` puts at the user's disposal the two METAPOST variables '`w`' and '`h`' with values 23 times and 45 times the usual LaTeX parameter `\unitlength`. Passing to METAPOST the `\linewidth` is pretty complicated, since it requires to divide the `\linewidth` by the unit length. Moreover, only two parameters are too few.

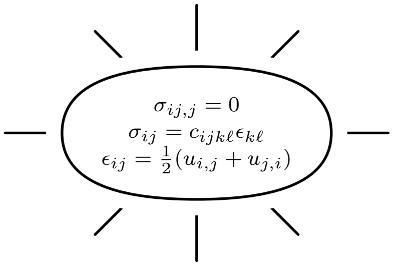
Therefore I chose to write the METAPOST files expanding commands. This works since METAPOST instructions have no backslashes in them and the command `\mpdim` used for passing parameters from the current document is especially tailored in order that its expansion is accepted by METAPOST. With the obvious exception of things inside `btex...etex`. The solution exploits the `\unexpanded` command made available by the modern TeX engines.

The construction `\btex...etex` is equivalent to write

```
btex \unexpanded{...} etex
```

Figure 6 An example from Alan Hoenig's book

```
\begin{mpost}[mpxprogram=latex,mpsettings={input boxes;}]
u:=1pc; path p[]; p0=(-6u,0) -- (6u,0);
p1=(0,-4u)--(0,4u);
p2=(0,-4.5u)--(0,4.5u); p2:=p2 rotated -45;
circleit mech(\btex
{\fontsize{8}{10}\selectfont
$\begin{array}{c}
\sigma_{ij,j}=0\\
\sigma_{ij}=c_{ijk\ell}\epsilon_{k\ell}\\
\epsilon_{ij}=\frac{1}{2}(u_{i,j}+u_{j,i})
\end{array}$}
etex);
mech.dx=mech.dy; mech.c=origin;
pickup pencircle scaled 1pt;
draw p0; draw p1; draw p2; draw p2 rotated 90;
pickup pencircle scaled u; unfilldraw bpath.mech;
pickup pencircle scaled 1pt; drawboxed(mech);
\end{mpost}
```


$\sigma_{ij,j} = 0$
$\sigma_{ij} = c_{ijk\ell}\epsilon_{k\ell}$
$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$

and what is written to the METAPOST file is precisely

```
btex ... etex
```

with no expansion of commands.

There is another catch: for TeXnical reasons, environments in `\btex...etex` parts cannot be used without protection. This is because the METAPOST input is gathered as an argument delimited by `\end`. The solution is to put the inner environments inside braces. An example, taken from Hoenig [2, p. 416], is found in figure 6, where the input is just like Hoenig's except for a minor correction (`3.5u` in the original should be `4.5u`) and having put everything inside `\btex...etex` in braces (and writing `\btex` instead of `btex`, of course).

Another way to cope with this would be to write simply `\array` instead of `\begin{array}` and `\endarray` instead of `\end{array}`, which is how the `mpost` environment for this example has actually been written.

9 Definitions using the `mpost` environment

Some examples have already been presented, but it's best to underline again that the environment can go inside the replacement text of a definition; don't use `\mpost` and `\endmpost`, though, just `\begin{mpost}` and `\end{mpost}`. Commands defined this way are *fragile*: don't use them in moving arguments. As an example, we can use a variation on a drawing from [2]:

```
\newcommand{\squares}[2][black]{%
  \begin{mpost}
  boolean timetofillbox;
  sfactor:=.95;
  path p,q; u:=\mpdim{#2\textwidth}/2;
  p := (u,u) -- (-u,u) -- (-u,-u) -- (u,-u) -- cycle;
  theta := 0; dtheta := .005;
  forever:
    theta := theta+dtheta;
    q := p scaled sfactor rotated theta;
    exitif ypart(point 0 of q) > u;
  endfor;
  fill p withrgbcolor #1; timetofillbox:=false;
  for i:=1 upto 51:
    p := p scaled sfactor rotated theta;
    if timetofillbox:
      fill p withrgbcolor #1; timetofillbox:=false;
    else:
      fill p withrgbcolor white-#1; timetofillbox:=true;
    fi
  endfor
  \end{mpost}}
```

The optional argument is an RGB color, either with a predefined name such as `black` (default) or a triple such as `(.165,.325,.553)`; the second argument is the size of the square, expressed as a fraction of `\textwidth`, for example `0.8`. In figure 7 you get three variations input as

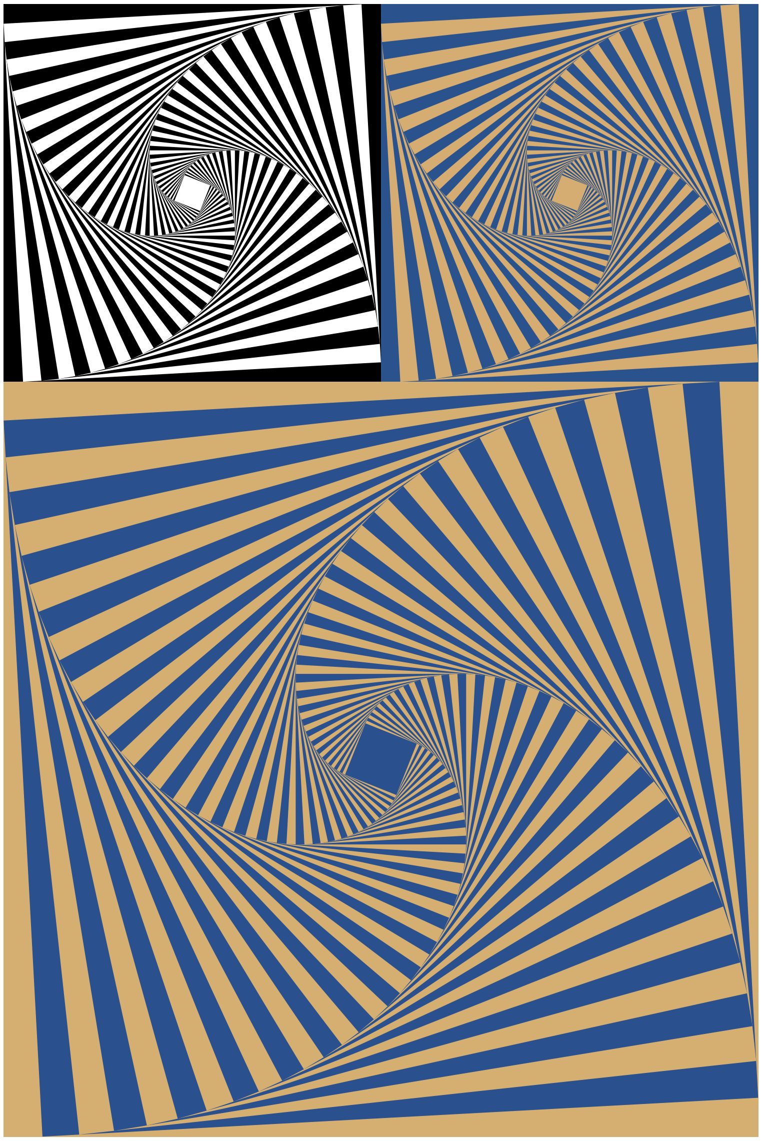
```
\begin{minipage}{\textwidth}
\centering\offinterlineskip
\squares{.4}\squares[(.165,.325,.553)]{.4}\\
\squares[(.835,.685,.447)]{.8}
\end{minipage}
```

that also show how `gmp` uses the hi-res bounding box provided by METAPOST.

10 The `mpost*` environment

No command in the body of the `mpost*` environment will be interpreted in any special way, so don't use them except than encapsulated into `btex...etex`. Here there's no need to write `\btex`, but it doesn't hurt either. It's however necessary

Figure 7 Squares taken from Hoenig's book



to hide inside braces the LaTeX command `\end` like for the sibling environment `mpost`. It's possible to pass variables to the picture from the LaTeX context by exploiting the option `mpsettings`: for example, with a prologue such as

```
\begin{mpost*}[mpsettings{u:=\mpdim{\unitlength}}]
```

you'll be able to use in the METAPOST code the variable u which will be set to the current `\unitlength` (don't say `save u;` in the METAPOST code, of course).

The `mpost*` environment *can* go inside the argument to a command.

11 XeLaTeX and gmp

The present package is compatible with XeLaTeX; in this case it performs also a conversion to PDF of the MPS file for inclusion. There are some limitations, though: it's impossible to typeset labels to pictures with system fonts, as it would be in the main document. Indeed METAPOST can't use XeTeX for typesetting the labels, only LaTeX; this limitation is in METAPOST, we hope that in the future it will be removed. If your main font is Linux Libertine, for instance, you can typeset labels by saying

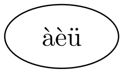
```
\usempxpackage[utf8]{inputenc}
\usempxpackage{libertine}
```

(possibly passing some option to the latter package). Many fonts that you can use with XeLaTeX can also be used in LaTeX, of course without access to the full character set.

12 Examples

Let's try some weird input with foreign characters.

```
\usepackage[latin1]{inputenc}
...
\usempxpackage[latin1]{inputenc}
...
\begin{center}
\begin{mpost}[mpxprogram=latex]
label(\btex àèü etex, origin);
draw fullcircle xscaled 1.5cm yscaled 2\mpdim\baselineskip;
\end{mpost}
\end{center}
```



Always use the same option to inputenc as your main file or, if you are working with XeLaTeX,

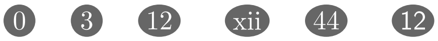
```
\usempxpackage[utf8]{inputenc}
```

in your document's preamble. Also `utf8x` is acceptable, of course, in this case.

Let's see instead a case where expansion in a `btex...etex` construction is needed: we want to enclose numbers, colored in white, inside an elliptical gray background. The vertical axis of the ellipse should be as high as the baseline skip and it should be a circle if the number has only one digit; the additional problem is that the number can be given as the representation of some counter (for example `enumi`, a possible application is evident).

```
\newcounter{mycount}
\newlength{\mylen}
\newcommand{\circleit}[2][]{\settowidth{\mylen}{#2}%
  \begin{mpost}[mpxprogram=latex]
    fill fullcircle
      xscaled max(\mpdim{\mylen+6pt},\mpdim{\baselineskip})
      yscaled \mpdim{\baselineskip}
      withcolor .4white;
    draw thelabel(btex \unexpanded{#1}#2 etex, origin)
      withcolor white;
  \end{mpost}}

\circleit{\themycount}\quad
\setcounter{mycount}{3}\circleit{\themycount}\quad
\setcounter{mycount}{12}\circleit{\themycount}\quad
\renewcommand{\themycount}{\roman{mycount}}%
\circleit{\themycount}\quad
\circleit{44}\quad
\renewcommand{\themycount}{\arabic{mycount}}%
\circleit[\sffamily]{\themycount}
```



In this case we *want* that what is inside `btex...etex` is expanded, in case it is `\themycount`. Of course we have to be sure that the expansion of the command consists only of characters. So, for example,

```
\renewcommand{\themycount}{\textsf{\arabic{mycount}}}
```

would not work for the last example. This is the reason for having defined an optional argument to `\circleit`. The trick of using this argument in `\unexpanded` keeps every program happy and the label is typeset correctly. More fun for the TeXackers!

13 Conversion from emp

It's quite easy to convert old files that use `emp` without changing too much; a typical `emp` environment is

```
\begin{emp}[xyz](15,30)
...;
\end{emp}
```

where the dots stand for METAPOST statements and `xyz` is the name for subsequent use. This may be transformed into

```
\begin{mpost}[name=xyz,use]
w:=\mpdim{15\unitlength};
h:=\mpdim{30\unitlength};
...;
\end{mpost}
```

An `empgraph` environment such as

```
\begin{empgraph}[xyz](15,30)
...;
\end{empgraph}
```

must be transformed into

```
\begin{mpost}[name=xyz,use](15,30)
w:=\mpdim{15\unitlength};
h:=\mpdim{30\unitlength};
draw begingraph(w,h);
...;
endgraph;
\end{mpost}
```

Remember to change all `btex` into `\btex` and the auxiliary commands into the similar ones of `gmp`: it should be almost straightforward. If people ask for it, I might add a compatibility layer for `emp`.

Alternatively, the first case might be treated as

```
\begin{mpost*}[name=xyz,
  mpsettings={w:=\mpdim{15\unitlength};
    h:=\mpdim{30\unitlength};}]
...;
\end{mpost*}
```

You might prefer this strategy if you have many `btex...etex` pairs. Look out for `\end` commands, though: if not properly braced, they will confuse the processing of the environment's content.

14 Implementation

Before doing anything, we check that the engine knows about `\unexpanded` and `\dimexpr`; otherwise we say good-bye.

```
\@ifundefined{eTeXversion}{%
  \PackageError{gmp}{This package requires a e-TeX extensions}
    {The gmp package requires a modern TeX engine, please upgrade}%
  \endinput}{}
```

We need xkeyval for its key-value interface. Then we define some conditional for later usage and the package options.

```
\RequirePackage{xkeyval}[2005/01/30]

\newif\ifgmp@latex % true if always using latex for mpx
\newif\ifgmp@locallatex % true if using latex for mpx for a
                        % single figure

\newif\ifgmp@nogen % true if we are not generating the mps files
\newif\ifgmp@nowrite % true if we are not writing the mp files
\newif\ifgmp@warn % true if we are issuing the final message

\DeclareOptionX<ggmp>{tex}{\def\gmp@mpxprogram{tex}%
  \gmp@latexfalse\gmp@locallatexfalse}
\DeclareOptionX<ggmp>{latex}{\def\gmp@mpxprogram{latex}%
  \gmp@latextrue\gmp@locallatextrue}

\DeclareOptionX<ggmp>{noshellescape}{\gmp@nogentrue}
\DeclareOptionX<ggmp>{shellescape}{\gmp@nogenfalse}

\DeclareOptionX<ggmp>{nowrite}{\gmp@nowritetrue}
\DeclareOptionX<ggmp>{write}{\gmp@nowritefalse}

\DeclareOptionX<ggmp>{envname}{\def\gmp@envname{#1}}
\DeclareOptionX<ggmp>{extension}{\def\gmp@ext{#1}}

\def\gmp@preoptions{}
\DeclareOptionX<ggmp>{everymp}{\edef\gmp@preoptions{\gmp@preoptions^^J#1}}

\DeclareOptionX<ggmp>{clean}{\def\gmp@clean{#1}}

\DeclareOptionX<ggmp>{rmcommand}{\def\gmp@remove{#1}}
\DeclareOptionX<ggmp>{postrmcommand}{\def\gmp@postremove{#1}}

\def\gmp@jobname{\jobname}
```

Now we set the default options: the mpx program is TeX; we write out the auxiliary files and do not assume shell-escape; no cleaning after compilation of the METAPOST files; default environment names are `mpost` and `mpost*`; default mp-extension is `+mp`. Then we read the package options.

```
\ExecuteOptionsX<ggmp>{tex,noshellescape,write,clean=none}
\ExecuteOptionsX<ggmp>{envname=mpost,extension=+mp,
  rmcommand=rm -f,postrmcommand={}}
\ProcessOptionsX<ggmp>\relax
```

We need some auxiliary packages, they'll probably be loaded anyway, with the exception of environ that's employed for the `mpost*` environment, but has no known incompatibility.

```
\RequirePackage{graphicx}
\RequirePackage{ifpdf}
```

```
\RequirePackage{ifxetex}
\RequirePackage{environ}
```

With XeLaTeX we need to run epstopdf; we don’t support LuaLaTeX, because looking at shell-escape is useless, at the moment, even via Heiko Oberdiek’s pdftexcmds (at least for the time being).

```
\ifxetex
  \let\gmp@shellescape\shellescape
  \def\gmp@preoptions{prologues:=3;^^J}
\else
  \ifdefined\pdfshellescape % (pdf)latex
    \let\gmp@shellescape\pdfshellescape
  \else % lualatex
    \chardef\gmp@shellescape=\z@
    \gmp@nogentrue
  \fi
\fi
```

We define some warning messages; in some cases the message enables or disables other messages.

```
\def\gmp@msgdisallowed{\PackageWarningNoLine{gmp}{Compilation and
  writing of MetaPost files has been\MessageBreak disallowed by the
  ‘nowrite’ option}\gdef\gmp@nemessage{}}
\def\gmp@msgnonexistent{\PackageWarning{gmp}{Non existent MetaPost
  file requested}\ifgmp@warn\else\global\gmp@warntrue\fi}
\def\gmp@msgrequestx{\PackageWarningNoLine{gmp}{The MetaPost file you
  requested does not exist,\MessageBreak perhaps by a compilation
  error}}
\def\gmp@msgremember{\PackageWarningNoLine{gmp}{Remember to run ‘sh
  \gmp@jobname\gmp@ext.sh’ and rerun (pdf)LaTeX}}
```

The following lines are to avoid spurious messages, opening and writing of files if the user has not activated the shell escape feature but has given the shell-escape option.

```
\def\gmp@msgensure{%
  \PackageWarningNoLine{gmp}{Ensure that you have enabled the shell
    escape feature, or\MessageBreak you can be in trouble. The
    available MetaPost generated\MessageBreak files will be used
    anyway. Use option ‘nowrite’ if they\MessageBreak are already
    in final form}%
  \let\gmp@message\@gobble
  \let\gmp@writexviii\@gobble
  \def\gmp@openout##1##2\@nil{}%
  \let\gmp@write\@gobbletwo
  \let\gmp@closeout\@gobble}
```

Now we define some abbreviations for primitive actions, so that we are able to change their meaning at will.

```
\let\gmp@message\message
\def\gmp@writexviii{\immediate\write18 }
\def\gmp@openout#1#2\@nil{\immediate\openout#1#2}
```

```
\def\gmp@write{\immediate\write}
\def\gmp@closeout{\immediate\closeout}
```

`\gmp@command`

Here we define the main internal macro, `\gmp@command`, which has different definitions depending on the package options. If the `nowrite` option is valid, there's even no need for it; otherwise it must behave differently: if we are doing immediate compilation via shell-escape it must act by emitting `\gmp@writexviii`, otherwise it must write the command in the batch file to be run after the LaTeX run. In the latter case we also open an output stream and write a prologue on it.

```
\def\gmp@setupmacros{%
\ifgmp@nowrite
  \let\gmp@nemessage\gmp@msgdisallowed
\else
  \ifgmp@nogen
    \let\gmp@nemessage\gmp@msgnonexistent
    \def\gmp@command{%
        mpost \ifx\gmp@mpmem\empty\else\space\gmp@mpmem\space\fi}%
    \newwrite\gmp@shellout
    \immediate\openout\gmp@shellout=\gmp@jobname\gmp@ext.sh%
    \immediate\write\gmp@shellout{\string##!/bin/sh}%
    \def\gmp@shellcommand##1{\immediate\write\gmp@shellout{##1}}%
    \AtEndDocument{\ifgmp@warn\gmp@msgremember\gmp@warnfalse\fi}%
  \else
    \let\gmp@nemessage\gmp@msgrequestx
    \ifnum\gmp@shellescape=\@ne
      \def\gmp@command{%
        mpost -interaction=nonstopmode %
        \ifx\gmp@mpmem\empty\else\space\gmp@mpmem\space\fi}%
      \def\gmp@shellcommand##1{%
        \gmp@message{^^J(gmp) Doing external command^^J(gmp) \string"}%
        \gmp@writexviii{echo ##1'\string"'}\gmp@writexviii{##1}%
      }%
    \else
      \gmp@msgensure\let\gmp@shellcommand\@gobble
    \fi
  \fi
\fi
}
\gmp@setupmacros
```

`\gmpoptions`

Next we define `\gmpoptions` which can be used to set package options in a more friendly way. However we disallow it outside the preamble, because otherwise the status of the macros would not be predictable.

```
\def\gmpoptions#1{\setkeys{ggmp}{#1}\gmp@setupmacros}
\@onlypreamble\gmpoptions
```

We next define a standard LaTeX preamble for mpx.

```
\def\gmp@latexpreamble{%^^J%
  \gmp@percent\string&latex^^J%
  \string\documentclass[1\@ptsize pt]{article}^^J%
  \ifx\empty\gmp@packages\else\the\gmp@packages^^J\fi%
  \ifx\empty\gmp@commands\else\the\gmp@commands^^J\fi%
  \string\begin{document}^^J}
```

`\usempxclass` `\usempxpackage`

If one wants a non standard LaTeX class, the command `\usempxclass`, with the same syntax as `\documentclass`, is provided. Similarly for `\usempxpackage`; `\resetmpxpackages` clears out the loading of packages.

```
\newcommand\usempxclass[2][]{%
  \def\gmp@latexpreamble{^^J%
    \gmp@percent\string&latex^^J%
    \string\documentclass[#1]{#2}^^J%
    \ifx\empty\gmp@packages\else\the\gmp@packages^^J\fi%
    \ifx\empty\gmp@commands\else\the\gmp@commands^^J\fi%
    \string\begin{document}^^J}}
\newtoks\gmp@packages
\newcommand\usempxpackage[2][]{%
  \gmp@packages=\expandafter{\the\gmp@packages^^J%
    \usepackage[#1]{#2}}}
\newcommand\resetmpxpackages{\gmp@packages={}}
```

`\mpxcommands`

With `\mpxcommands` one can insert definitions to be used for METAPOST label composition: these are seen only if LaTeX is used for mpx. There's also `\resetmpxcommands` in case of need.

```
\newtoks\gmp@commands
\newcommand\mpxcommands[1]{%
  \gmp@commands=\expandafter{\the\gmp@commands^^J%
    #1}}
\newcommand\resetmpxcommands{\gmp@commands={}}
```

`\mpdim`

The `\mpdim` macro is used in the `mpost` environment or in the value of the key `mpsettings` for passing length depending on current conditions. We exploit `\dimexpr` that strips out possible 'elastic' components and allows for doing calculations. The result is written out surrounded by `begingroup` and `endgroup` that make the number obtained look like a variable at METAPOST's eyes.

```
\def\mpdim#1{ begingroup \the\dimexpr#1\relax\space endgroup }
```

Now we start the hard task of writing out the auxiliary files. So we define an output stream and a counter that keeps track of the figure names; in order to know the total number of the pictures, at end-of-document we define an internal label which is currently unused, though.

```
\newwrite\gmp@out
\newcounter{gmp@count}
\AtEndDocument{%
  \refstepcounter{gmp@count}%
  \label{gmp@finallabel}%
}
```

`mpost` The `mpost` environment (but the name can be set at package loading) must read the 'local options' given as an optional argument as the usual list of key-value statements. First of all it must step the METAPOST figure counter and set a label for it for subsequent calling of the graphics, then we grab the option given to the environment.

```
\newenvironment{\gmp@envname}[1][]
  {\@bsphack
   \global\gmp@usefalse
   \refstepcounter{gmp@count}\label{gmp@label@\thegmp@count}%
   \bgroup
   \edef\gmp@number{\thegmp@count}%
   \gmp@grab{#1}}
  {\@esphack}
```

`mpost*` The `mpost*` environment writes out its contents without TeX macro expansion; we simply use `mpost`: so we gather the optional argument into a scratch token register, collect the body of the environment and pass it to the `\gmp@writeunex` macro which calls `\begin{mpost}` (possibly with the name chosen by the user), to which are passed the options. The first `\unexpanded` is removed by `\edef`, the second one by the writing process.

```
\def\gmp@writeunex#1{\begingroup
  \edef\x{\endgroup\noexpand\begin{\gmp@envname}[\the\toks@]%
    \unexpanded{\unexpanded{#1}}\noexpand\end{\gmp@envname}}\x}
\newenvironment{\gmp@envname*}[1][]
  {\toks@{#1}\Collect@Body\gmp@writeunex}{}
```

Here we define the key-value interface for the options to the environments, with some auxiliary macros and conditionals; `\gmp@fourdigits` is used for the output file names, we think that more than 9999 pictures are difficult to be present for a single file; but this will not limit at all the user's possibilities.

```
\def\gmp@fourdigits#1{%
  \ifnum#1<10   0\fi
  \ifnum#1<100  0\fi
  \ifnum#1<1000 0\fi\number #1}
\def\gmp@not@a@name@{@not@a@name@}
\def\gmp@choosetex{\gmp@locallatexfalse}
\def\gmp@chooselatex{\gmp@locallatextrue}
\def\gmp@doiflatex#1{\ifgmp@latex#1\else\ifgmp@locallatex#1\fi\fi}
\def\gmp@mpmem{}
\newif\ifgmp@use
\define@key{gmp}{name}[@not@a@name@]{%
  \def\gmp@thisname{#1}%
  \expandafter\xdef\csname gmp@fig#1\endcsname
    {\gmp@fourdigits{\gmp@number}}}
\define@key{gmp}{use}[true]{\global\csname gmp@use#1\endcsname}
\define@key{gmp}{mpmem}{\def\gmp@mpmem{-mem #1}}
\define@key{gmp}{mpsettings}{\edef\gmp@preoptions{\gmp@preoptions^^J#1}}
\define@key{gmp}{mpxcommands}{%
```

```
  \gmp@commands=\expandafter{\the\gmp@commands^^J#1}}
\define@key{gmp}{mpxprogram}{%
  \def\gmp@mpxprogram{#1}%
  \csname gmp@choose#1\endcsname}
\define@key{gmp}{runs}{\chardef\gmp@runs=#1\relax}
```

`\gmp@grab`

The `\gmp@grab` macro sets the value of the keys; the `name` must be initially empty and the number of runs should be 1. The conditional `\if@gmpuse` is set to false at the beginning of the environment, and to true when `use` is given or when there's no name. Finally the macro calls `\gmp@innermpost`.

```
\def\gmp@grab#1{%
  \setkeys{gmp}{name}%
  \setkeys{gmp}{runs=1}%
  \setkeys{gmp}{#1}%
  \ifx\gmp@thisname\gmp@not@a@name@
    \global\gmp@usetrue
  \fi
  \gmp@setup\gmp@innermpost}
```

Before defining the macro responsible for the task of writing out the METAPOST files, we do some housekeeping.

We ensure that the character we need are written correctly in the METAPOST files; so we define a 'printing' percent sign and change the category code of characters that might be made active by babel or other packages. The semicolon is made active: it will output a semicolon followed by a new line character; in the environment `\par` means 'go to a new line', so that empty lines in a `mpost` environment will cause a new line in the output file. However this doesn't work currently for `mpost` environments given as arguments to other commands; since the normal buffer size is around 50 000, this should not be a problem.

```
\begingroup\@makeother\%\def\x{\endgroup\def\gmp@percent{%}}\x
\def\gmp@activesc{\catcode`\;=\active
  \begingroup\lccode`\~=`\;
  \lowercase{\endgroup\edef~}{\string;^^J}}
\def\gmp@otherchars{\do\!\do\=\do\:\do\"\do\?\do\'\do\`\do\|}
```

`\gmp@setup`

The following macro `\gmp@setup` starts writing the output files, after setting category codes and something else; it doesn't write anything if the option `nowrite` is in force, otherwise writes a prologue and all needed preliminaries, up to `beginfig(`⟨*number*⟩`)`; this number is actually irrelevant, because we use `outputtemplate`, but it should correspond to the number appearing in the name of the output file.

```
\def\gmp@setup{%
  \let\do\@makeother\gmp@otherchars\newlinechar=`\^^J
  \gmp@activesc
  \def\par{^^J}%
  \ifgmp@nowrite\else
    \edef\@temp{\gmp@jobname\gmp@ext\gmp@fourdigits{\gmp@number}.mp}%
    \expandafter\gmp@openout\expandafter\gmp@out\@temp\@nil
    \gmp@write\gmp@out{%
      \gmp@percent\gmp@percent\space
        Do not edit, this file has been generated^^J%
      \gmp@percent\gmp@percent\space
        automatically by \jobname.tex via gmp.sty^^J^^J%
      \gmp@doiflatex{verbatimtex\gmp@latexpreamble etex;^^J}%
      \ifx\empty\gmp@preoptions\else\gmp@preoptions^^J\fi%
      outputtemplate:= "\gmp@percent j.mps";^^J%
      beginfig(\gmp@number);%^^J%
    }%
  \fi}
```

`\gmp@innermpost` This is the central macro which collects the body of the environment; we chose to use `\end` as delimiter so that the environment can be used inside other arguments or definitions; maybe it's not the best way to do this, but other than requiring protection of `\end` commands inside the environment seems to have no side effects. The environment's content is written out to the METAPOST file, followed by `endfig;` and `end`; if LaTeX is used for the labels, also the closing statements are added (though they are not strictly necessary). Then `\gmp@shellcommand` is performed one or more times according to the value of `runs`. It will run METAPOST stopping LaTeX momentarily or write the command to the shell script, according to the current package options. In the case of XeLaTeX it will also call or write for a run of `epstopdf`. Finally, it executes `\end{mpost}` and uses the object if it has been requested to do, that is, if `\ifgmp@use` is true.

```
\long\def\gmp@innermpost#1\end#2{%
  \ifgmp@nowrite\else
    \gmp@write\gmp@out{%
      #1^^Jendfig;^^Jend.%
      \gmp@doiflatex{^^Jverbatimtex^^J\string\end{document}^^Jetex}}%
    \gmp@closeout\gmp@out
    \count@=\gmp@runs
    \loop\ifnum\count@>\z@
      \gmp@shellcommand{\gmp@command\space
        -tex=\gmp@mpxprogram\space\gmp@jobname\gmp@ext
        \gmp@fourdigits{\gmp@number}}%
    \advance\count@\m@ne
    \repeat
    \ifxetex
      \gmp@shellcommand{epstopdf --hires \gmp@jobname\gmp@ext
        \gmp@fourdigits{\gmp@number}.mps}%
    \fi
  \fi
  \egroup
  \end{#2}\ifgmp@use
  \gmp@usempost{\thegmp@count}\fi}
```

`\usempost` The macros for using the compiled pictures are almost straightforward: the files will have extension `.mps` for LaTeX or pdfLaTeX and `.pdf` for XeLaTeX. We check that they exist and load them or issue a warning.

```
\ifxetex
  \def\gmp@usempost#1{%
    \edef\gmp@thempsfile{\gmp@jobname\gmp@ext\gmp@fourdigits{#1}}%
    \IfFileExists{\gmp@thempsfile.pdf}%
    {\includegraphics{\gmp@thempsfile.pdf}}%
    {\gmp@nemessage\gmp@box}}
  \newcommand\usempost[2][]{%
    \IfFileExists{\gmp@jobname\gmp@ext\csname gmp@fig#2\endcsname.pdf}%
    {\includegraphics[#1]
      {\gmp@jobname\gmp@ext\csname gmp@fig#2\endcsname.pdf}}%
    {\gmp@nemessage\gmp@box}}
\else
  \def\gmp@usempost#1{%
    \edef\gmp@thempsfile{\gmp@jobname\gmp@ext\gmp@fourdigits{#1}}%
    \IfFileExists{\gmp@thempsfile.mps}%
    {\includegraphics[hiresbb]{\gmp@thempsfile.mps}}%
    {\gmp@nemessage\gmp@box}}
\newcommand\usempost[2][]{%
  \IfFileExists{\gmp@jobname\gmp@ext\csname gmp@fig#2\endcsname.mps}%
  {\includegraphics[hiresbb,#1]
    {\gmp@jobname\gmp@ext\csname gmp@fig#2\endcsname.mps}}%
  {\gmp@nemessage\gmp@box}}
\fi
\def\gmp@box{\fbox{\@ifundefined{color}{}{\color{red}}MP}}
```

Now we clean up our directory from the auxiliary files, at the user's request, of course. The `clean` keyword may be specified in the package options or in `\gmpoptions` to have a value among `none`, `aux`, or `mp`.

At end document we look at the value of this key and do what's desired by the user. If the value is not recognized we issue a warning and act as if `clean=none` had been given; otherwise we execute the requested command or write them in the batch script.

```
\AtEndDocument{%
  \@ifundefined{gmp@doclean@\gmp@clean}
    {\expandafter\gmp@badcleaning\expandafter{\gmp@clean}}
    {\@nameuse{gmp@doclean@\gmp@clean}}}
\def\gmp@badcleaning#1{\PackageWarningNoLine{gmp}{%
  Wrong cleaning option ‘clean=#1’; ‘clean=none’ used}}
\def\gmp@doclean@none{}
\def\gmp@doclean@aux{%
  \gmp@shellcommand{\gmp@remove\space
    \jobname\gmp@ext*.log\space\gmp@postremove}%
  \gmp@shellcommand{\gmp@remove\space
    \jobname\gmp@ext*.mpx\space\gmp@postremove}%
  \gmp@shellcommand{\gmp@remove\space
    \jobname\gmp@ext*.mpo\space\gmp@postremove}%
    }
\def\gmp@doclean@mp{%
  \gmp@shellcommand{\gmp@remove\space
274     \jobname\gmp@ext*.log\space\gmp@postremove}%
275   \gmp@shellcommand{\gmp@remove\space
276     \jobname\gmp@ext*.mpx\space\gmp@postremove}%
277   \gmp@shellcommand{\gmp@remove\space
278     \jobname\gmp@ext*.mpo\space\gmp@postremove}%
279   \gmp@shellcommand{\gmp@remove\space
280     \jobname\gmp@ext*.mp\space\gmp@postremove}%
281   \ifxetex
282     \gmp@shellcommand{\gmp@remove\space
283     \jobname\gmp@ext*.mps\space\gmp@postremove}%
284   \fi}
```

`\verbatimtex` `\btex` Final definition for not expanding macros in the METAPOST labels.

```
285 \long\def\verbatimtex#1etex{verbatimtex \unexpanded{#1} etex}
286 \long\def\btex#1etex{btex \unexpanded{#1} etex}
```

References

[1] Hans Hagen, Metafun, `http://www.pragma-ade.nl/general/manuals/metafun-p.pdf`

[2] Alan Hoenig, *TEX unbound*, Oxford University Press, Oxford and New York, 1998.

[3] Steve Peter, "`\starttext`, Swelled Rules and METAPOST", The PracTEX Journal, n. 4 (2005).

Change History

v1.0
General: Initial version 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.