The graphicx package*

D. P. Carlisle S. P. Q. Rahtz

1999/02/16

1 Introduction

This package provides an alternative interface to the LaTeX 2_ε graphics functions. The command names provided are the same as in the standard package, and they use the same internal functions. However the meaning of the optional arguments is different. Note *only* the optional arguments have changed: any document which only uses the graphics commands with the mandatory arguments and/or the star-forms will work identically (with essentially identical implementation) with the two packages.

2 Key=Value Interface

When the decision to produce LaTeX 2_ε was made, certain 'guiding principles' were made (and published in the original announcement). One of these was that all new features would 'conform to the conventions of version 2.09'. Specifically this meant that new commands would obey the same basic syntax rules for arguments as the existing commands.

Standard LaTeX optional arguments are *positional*. If a command were to take three optional arguments, then there would be no way of specifying only the third, one would have to give all three, even if the first two were repeats of the default values. Basically this means that 'standard' optional arguments are not suitable if there is more than one option. Various existing packages (for LaTeX 2.09) have recognised this, and used 'named arguments' in various forms. Perhaps the two most noticeable are `psfig` and `pstricks`. With 'named arguments' (sometimes called 'attributes') each option is not tied to a particular position, but rather given a name (or key) and any options that must be set are set by explicitly associating this name with the desired value.

The members of the LaTeX3 project do appreciate the importance of this kind of syntax, but felt that rather than extending the syntax of LaTeX in an uncoordinated way, it would be better to keep with 'standard arguments' in LaTeX 2_ε, which is intended as a 'consolidation of existing LaTeX variants'. The long term planning for an eventual LaTeX3 release will then be able to consider the whole LaTeX user interface, and a suitable syntax for named arguments. It is important that such an interface design is not hampered by having to retain compatibility with earlier attempts at a named argument syntax. For this reason this `graphicx` package, which uses the named argument mechanism from the `keyval` package should be considered 'non standard' although it is supported by the same mechanism, and same authors as the 'standard' `graphics` package.

3 The User Interface

`\includegraphics` `*[`⟨*key-val list*⟩`]{`⟨*file*⟩`}`

`\includegraphics` `*[`⟨*llx,lly*⟩`][`⟨*urx,ury*⟩`]{`⟨*file*⟩`}`

*This file has version number v1.0f, last revised 1999/02/16.

Include a graphics file.

The star form is just for compatibility with the standard interface, and essentially just adds `clip` to the keys specified. Similarly the second, two-optional argument form is for increased compatibility with the standard package. The two optional argument form is not needed in the `keyval` interface.

Various 'keys' or named arguments are supported.

bb Set the bounding box. The argument should be four dimensions, separated by spaces.

bbllx,bblly,bburx,bbury Set the bounding box. Mainly for compatibility with older packages. `bbllx=a,bblly=b,bburx=c,bbury=d` is equivalent to `bb = a b c d`.

natwidth,natheight Again an alternative to `bb`. `natheight=h,natwidth=w` is equivalent to `bb = 0 0 h w`.

viewport Modify the bounding box specified in the file. The four values specify a bounding box *relative* to the `llx`,`lly` coordinate of the original box.

trim Modify the bounding box specified in the file. The four values specify the amounts to remove from the left, bottom, right and top of the original box.

hiresbb Boolean valued key. Defaults to `true`. Causes TEX to look for `%%HiResBoundingBox` comments rather than the standard `%%BoundingBox`. May be set to `false` to override a default setting of true specified by the `hiresbb` package option.

angle Rotation angle.

origin Rotation origin (see `\rotatebox`, below).

width Required width, a dimension (default units `bp`). The graphic will be scaled to make the width the specified dimension.

height Required height. a dimension (default units `bp`).

totalheight Required totalheight (ie height + depth). a dimension (default units `bp`). Most useful after a rotation (when the height might be zero).

keepaspectratio Boolean valued key (like `clip`). If it is set to true, modify the meaning of the `width` and `height` (and `totalheight`) keys such that if both are specified then rather than distort the figure the figure is scaled such that neither dimension *exceeds* the stated dimensions.

scale Scale factor.

clip Either 'true' or 'false' (or no value, which is equivalent to 'true'). Clip the graphic to the bounding box (or viewport if one is specified).

draft a boolean valued key, like 'clip'. locally switches to draft mode, ie. do not include the graphic, but leave the correct space, and print the filename.

type Specify the file type. (Normally determined from the file extension.)

ext Specify the file extension. *Only* for use with `type`.

read Specify the 'read file' which is used for determining the size of the graphic. *Only* for use with `type`.

command Specify the file command. *Only* for use with `type`.

The arguments are interpreted left to right. `clip`, `draft`, `bb`,, and `bbllx` etc. have the same effect wherever they appear. but the scaling and rotation keys interact.

Any scaling that is specified *before* rotation, is handled by the internal graphics inclusion function. Rotation, or any later scaling is handled by implicitly calling `\rotatebox` or `\scalebox`. So `[height=1in,angle=90]` scales the graphic to 1in, then rotates it, so it is one inch wide. `[angle=90,height=1in]` first rotates, then scales the result so that it is 1in high. A driver that can scale included graphics, but not arbitrary text will not be able to support the second form, as it will require a call to `\scalebox`, but the first form should work as there the scaling is handled by `\Ginclude@graphics`.

`\rotatebox` `[`⟨*key-val list*⟩`]{`⟨*angle*⟩`}{`⟨*text*⟩`}`

Rotate *text*.

The keys supported by `\rotatebox` are:

origin Specify the centre of rotation. `origin=`⟨*label*⟩, where the labels are up to two of `lrctbB` (`B` denotes the baseline, as for PSTricks).

x,y An alternative to `origin`. `x=`⟨*dimen*⟩`,y=`⟨*dimen*⟩ The x, y coordinate of the centre of rotation. As usual `\height` etc may be used.

units Specify the units used in the main argument. eg `units=-360` would mean that the argument refered to degrees *clockwise* instead of the default anti-clockwise rotation.

As an example `\rotatebox[origin=c]{180}{text}` will rotate "text" around its centre, thus creating a final box of the same dimensions as the original box. This is to be contrasted to the default behaviour, which rotates around the reference point on the baseline, thus producing a box that is mainly *below* the baseline.

4 Implementation

```
1 ⟨*package⟩
```

One new option is handled by keyval. It suppresses the error normally generated if an unknow keyval key is used. (This helps porting between drivers that use extended interfaces.)

```
2 \DeclareOption{unknownkeysallowed}
3   {\PassOptionsToPackage\CurrentOption{keyval}}
```

All other options are handled by the graphics package.

```
4 \DeclareOption*{\PassOptionsToPackage\CurrentOption{graphics}}
5 \ProcessOptions
```

This package requires these two building blocks.

```
6 \RequirePackage{keyval,graphics}
```

4.1 Graphics Inclusion

First we declare the 'bounding box' keys. These all use `\Gin@defaultbp` so that the ⟨*value*⟩ can be given as a length in the usual TEX units such as `cm` or as an integer, taken as `bp`.

`\KV@Gin@bb`

```
7 \define@key{Gin}{bb}
8           {\Gin@bboxtrue\Gread@parse@bb#1 \\}
```

`\KV@Gin@bbllx`
`\KV@Gin@bblly`
`\KV@Gin@bburx`
`\KV@Gin@bbury`

```
9 \define@key{Gin}{bbllx}
10          {\Gin@bboxtrue\Gin@defaultbp\Gin@llx{#1}}
11 \define@key{Gin}{bblly}
12          {\Gin@bboxtrue\Gin@defaultbp\Gin@lly{#1}}
```

```
\define@key{Gin}{bburx}
           {\Gin@bboxtrue\Gin@defaultbp\Gin@urx{#1}}
\define@key{Gin}{bbury}
           {\Gin@bboxtrue\Gin@defaultbp\Gin@ury{#1}}
```

`\KV@Gin@hiresbb` If set to true (the default) TEX will look for bounding box comments of the form `%%HiResBoundingBox` (which typically have real values) instead of the standard `%%BoundingBox` (which should have integer values). It may be set to false to override a package option of hiresbb.

```
\define@key{Gin}{hiresbb}[true]{%
  \edef\Gread@BBox{%
    \@percentchar\@percentchar
    \csname if#1\endcsname HiRes\fi
    BoundingBox}}
```

`\KV@Gin@natheight`

`\KV@Gin@natheight`

```
\let\KV@Gin@natwidth\KV@Gin@bburx
\let\KV@Gin@natheight\KV@Gin@bbury
```

`\KV@Gin@viewport` `\KV@Gin@trim` A 'viewport' is a user-specified area of the graphic to be included. It should not be confused with the 'Bounding Box' of a PS file. In fact, the origin for a viewport specification is the (llx,lly) lower left coordinate of the bounding box. If a viewport is specified, and clipping is turned on, clipping is based on the viewport, not on the boundingbox.

Both 'viewport' and 'trim' were suggested (and originally, but differently, implemented) by Arthur Ogawa.

```
\define@key{Gin}{viewport}
           {\let\Gin@viewport@code\Gin@viewport\Gread@parse@vp#1 \\}
\define@key{Gin}{trim}
           {\let\Gin@viewport@code\Gin@trim\Gread@parse@vp#1 \\}
```

`\Gread@parse@vp` Grabs four bounding box values like `\Gread@parse@bp` but saves them in alternative macros that are used in the viewport and trim cases to modify the bounding box read from the file.

```
\def\Gread@parse@vp#1 #2 #3 #4 #5\\{%
  \Gin@defaultbp\Gin@vllx{#1}%
  \Gin@defaultbp\Gin@vlly{#2}%
  \Gin@defaultbp\Gin@vurx{#3}%
  \Gin@defaultbp\Gin@vury{#4}}%
```

`\Gin@viewport` If a viewport is specified, reset the bounding box cordinates by adding the original origin, `\Gin@llx`, `\Gin@lly` to the new values specified as the viewport. The original Bounding box coordinates are saved in `\Gin@ollx`... some drivers might need this information (currently just `tcidvi`).

```
\def\Gin@viewport{%
  \let\Gin@ollx\Gin@llx
  \let\Gin@olly\Gin@lly
  \let\Gin@ourx\Gin@urx
  \let\Gin@oury\Gin@ury
  \dimen@\Gin@llx\p@\advance\dimen@ \Gin@vurx\p@
                       \edef\Gin@urx{\strip@pt\dimen@}%
  \dimen@\Gin@lly\p@\advance\dimen@ \Gin@vury\p@
                       \edef\Gin@ury{\strip@pt\dimen@}%
  \dimen@\Gin@llx\p@\advance\dimen@ \Gin@vllx\p@
                       \edef\Gin@llx{\strip@pt\dimen@}%
  \dimen@\Gin@lly\p@\advance\dimen@ \Gin@vlly\p@
                       \edef\Gin@lly{\strip@pt\dimen@}}
```

`\Gin@trim` If a trim is specified, reset the bounding box cordinates by trimming the four specified values off each side of the graphic.

```
\def\Gin@trim{%
  \let\Gin@ollx\Gin@llx
  \let\Gin@olly\Gin@lly
  \let\Gin@ourx\Gin@urx
  \let\Gin@oury\Gin@ury
  \dimen@\Gin@llx\p@\advance\dimen@ \Gin@vllx\p@
                       \edef\Gin@llx{\strip@pt\dimen@}%
  \dimen@\Gin@lly\p@\advance\dimen@ \Gin@vlly\p@
                       \edef\Gin@lly{\strip@pt\dimen@}%
  \dimen@\Gin@urx\p@\advance\dimen@ -\Gin@vurx\p@
                       \edef\Gin@urx{\strip@pt\dimen@}%
  \dimen@\Gin@ury\p@\advance\dimen@ -\Gin@vury\p@
                       \edef\Gin@ury{\strip@pt\dimen@}}
```

`\Gin@vllx` `\Gin@vlly` `\Gin@vurx` `\Gin@vury`

Four macros to hold the modifiers for the bounding box for viewport and trim specifications.

```
\let\Gin@vllx\Gin@llx\let\Gin@vlly\Gin@llx
\let\Gin@vurx\Gin@llx\let\Gin@vury\Gin@llx
```

`\KV@Gin@angle`

Specify a rotation. This is just handled by wrapping the `\includegraphics` command in a call to the internal version of `\rotatebox`. Normally this is the 'standard' version but if an `origin` key is used in `\includegraphics` then the *keyval* version of origin is used, and the `origin` key is passed on.

```
\define@key{Gin}{angle}
           {\Gin@esetsize
            \@tempswatrue
            \edef\@tempa{\toks@{\noexpand\Gin@erotate{#1}{\the\toks@}}}%
            \@tempa}
```

`\KV@Gin@origin`

Pass the origin key value on to `\rotatebox`. `\Gin@erotate` is initialised to `\Grot@box@std` later in the file, after the latter has been defined.

```
\define@key{Gin}{origin}[c]{%
  \def\Gin@erotate{\Grot@box@kv[origin=#1]}}
```

`\KV@Gin@width` `\KV@Gin@height`

Save the required height and width. The actual scaling is done later.

```
\define@key{Gin}{width}{\def\Gin@ewidth{#1}}
\define@key{Gin}{height}{\def\Gin@eheight{#1}}
```

`\KV@Gin@totalheight`

The same as `height` key, but locally changes `\Gin@eresize` to `\totalheight` from its default value of `\height`.

```
\define@key{Gin}{totalheight}{%
  \def\Gin@eresize{\totalheight}\def\Gin@eheight{#1}}
```

`\KV@Gin@keepaspectratio`

Boolean valued key (like `clip`). If it is set to true, modify the meaning of the `width` and `height` (and `totalheight`) keys such that if both are specified then rather than distort the figure the figure is scaled such that neither dimension *exceeds* the stated dimensions.

```
\define@key{Gin}{keepaspectratio}[true]{%
  \lowercase{\Gin@boolkey{#1}}{iso}}
```

`\KV@Gin@scale`

If the scaling is being handled externally, wrap `\includegraphics` in the internal form of `\scalebox`, otherwise locally define `\Gin@req@sizes` to calculate the required sizes based on scale factor.

```
\define@key{Gin}{scale}{%
  \if@tempswa
    \edef\@tempa{\toks@{\noexpand\Gscale@box{#1}[#1]{\the\toks@}}}%
    \@tempa
  \else
    \def\Gin@req@sizes{%
      \def\Gin@scalex{#1}\let\Gin@scaley\Gin@exclamation
      \Gin@req@height\Gin@scalex\Gin@nat@height
```

```
      \Gin@req@width\Gin@scalex\Gin@nat@width}%
  \fi
  \@tempswatrue}
```

`\KV@Gin@draft` Locally set the draft switch to true. This is used by the code in graphics package to suppress the file inclusion.

```
\define@key{Gin}{draft}[true]{%
  \lowercase{\Gin@boolkey{#1}}{draft}}
```

`\KV@Gin@clip` Locally set the clip switch to true. This is used by the code in graphics package to suppress the printing of anything outside the bounding box specified.

```
\define@key{Gin}{clip}[true]{%
  \lowercase{\Gin@boolkey{#1}}{clip}}
```

`\KV@Gin@type` If you use 'type' you must use no extension in the main argument and you must use 'ext'. You can also use 'read' and 'command'.

```
\define@key{Gin}{type}{%
  \def\Ginclude@graphics##1{%
    \begingroup
    \def\Gin@base{##1}%
    \edef\@tempa{{#1}{\Gin@eread}{\Gin@ecom{##1\Gin@eext}}}%
    \expandafter\Gin@setfile\@tempa
    \endgroup}}
```

`\KV@Gin@ext` Specify an extension, for use with the 'type' key.

```
\define@key{Gin}{ext}{\def\Gin@eext{#1}}
\let\Gin@eext\@empty
```

`\KV@Gin@read` Specify a read file, for use with the 'type' key. You may want to globally set this to `*` using `\setkeys`. `*` means read the graphic file for size info, as in `\DeclareGraphicsRule`.

```
\define@key{Gin}{read}{%
 \def\Gin@eread{#1}%
 \def\@tempa{*}\ifx\@tempa\Gin@eread\def\Gin@eread{\Gin@eext}\fi}
\let\Gin@eread\@empty
```

`\KV@Gin@command` Specify a command, for use with the 'type' key.

```
\define@key{Gin}{command}{\def\Gin@ecom##1{#1}}
\let\Gin@ecom\@firstofone
```

`\Gin@boolkey` Helper function for defining boolean valued functions. The order of arguments allows `\lowercase` to only act on the user-supplied argument.

```
\def\Gin@boolkey#1#2{%
  \csname Gin@#2\ifx\relax#1\relax true\else#1\fi\endcsname}
```

`\Gin@esetsize` Arrange for the final size to be set, either by wrapping the include graphics call in `\scalebox`, or by redefining `\Gin@req@sizes` appropriately.

```
\def\Gin@eresize{\height}
\def\Gin@esetsize{%
  \let\@tempa\Gin@exclamation
  \if@tempswa
```

External. Wrap the `\includegraphics` command in a call to the internal form of `\scalebox` to handle the rotation.

```
    \edef\@tempa{\toks@{\noexpand
              \Gscale@@box\noexpand\Gin@eresize
               {\Gin@ewidth}{\Gin@eheight}{\the\toks@}}}%
    \@tempa
  \else
```

Internal. Handle scaling with the `\includegraphics` command directly rather than calling `\scalebox`.

```
    \ifx\Gin@ewidth\@tempa
      \ifx\Gin@eheight\@tempa
```

No resizing.

```
      \else
```

Just height specified.

```
        \let\Gin@@eheight\Gin@eheight
        \def\Gin@req@sizes{%
          \Gscale@div\Gin@scaley\Gin@@eheight\Gin@nat@height
          \let\Gin@scalex\Gin@exclamation
          \setlength\Gin@req@height\Gin@@eheight
          \Gin@req@width\Gin@scaley\Gin@nat@width}%
      \fi
    \else
      \ifx\Gin@eheight\@tempa
```

Just width specified.

```
        \let\Gin@@ewidth\Gin@ewidth
        \def\Gin@req@sizes{%
          \Gscale@div\Gin@scalex\Gin@@ewidth\Gin@nat@width
          \let\Gin@scaley\Gin@exclamation
          \setlength\Gin@req@width\Gin@@ewidth
          \Gin@req@height\Gin@scalex\Gin@nat@height}%
      \else
```

Both height and width specified.

```
        \let\Gin@@ewidth\Gin@ewidth
        \let\Gin@@eheight\Gin@eheight
```

At this point can locally redefine `\Gin@nosize`. Instead of generating an error, just set the 'natural' size to be the 'requested size'. Previous versions of this package did not allow the use of `height` and `width` unless the natural size was known as otherwise LaTeX can not calculate the scale factor. However many drivers (especially for bitmap formats) can work this out themselves, so as long as both `height` and `width` are given, so LaTeX knows the size to leave, accept this. This assumes the code in the driver file will use the 'required height' information, not the scale factors, which will be set to 1!.

```
        \def\Gin@nosize##1{%
          \KV@Gin@natwidth\Gin@@ewidth
          \KV@Gin@natheight\Gin@@eheight}%
        \def\Gin@req@sizes{%
          \Gscale@div\Gin@scalex\Gin@@ewidth\Gin@nat@width
          \Gscale@div\Gin@scaley\Gin@@eheight\Gin@nat@height
```

Donald Arseneau requested this feature. If both `height` and `width` are chosen, choose the smaller scale factor rather than distort the graphic. This mode is turned on with the keepaspectratio key.

```
          \ifGin@iso
            \ifdim\Gin@scaley\p@>\Gin@scalex\p@
              \let\Gin@scaley\Gin@scalex
            \else
              \let\Gin@scalex\Gin@scaley
            \fi
          \fi
          \Gin@req@width\Gin@scalex\Gin@nat@width
          \Gin@req@height\Gin@scaley\Gin@nat@height}%
       \fi
     \fi
  \fi
  \let\Gin@ewidth\Gin@exclamation
  \let\Gin@eheight\Gin@ewidth}
```

`\Gin@req@height` The required final size.

`\Gin@req@width`

```
\newdimen\Gin@req@height
\newdimen\Gin@req@width
```

`\Gin@outer@scalex` Scale factors to pass to `\scalebox`.

`\Gin@outer@scaley`

```
\let\Gin@outer@scalex\relax
\let\Gin@outer@scaley\relax
```

`\Gin@angle` Rotation angle.

```
\let\Gin@angle\relax
```

`\Gin@ewidth` Final size, initialised for no scaling.

`\Gin@eheight`

```
\let\Gin@ewidth\Gin@exclamation
\let\Gin@eheight\Gin@ewidth
```

`\Gin@scalex` Scale factors. Initialised for no scaling.

`\Gin@scaley`

```
\def\Gin@scalex{1}
\let\Gin@scaley\Gin@exclamation
```

`\Gin@i` Use the same top level `\includegraphics` command as the standard interface. This will set the clipping switch, and then call `\Gin@i`.

```
\def\Gin@i{%
 \def\Gin@req@sizes{%
   \Gin@req@height\Gin@nat@height
   \Gin@req@width\Gin@nat@width}%
 \@ifnextchar[\Gin@ii{\Gin@ii[]}}
```

`\Gin@ii` Look for a second optional argument. If one optional argument is present, call `\setkeys` to process it,

```
\def\Gin@ii[#1]#2{%
    \def\@tempa{[}\def\@tempb{#2}%
    \ifx\@tempa\@tempb
      \def\@tempa{\Gin@iii[#1][}%
      \expandafter\@tempa
    \else
     \begingroup
       \@tempswafalse
       \toks@{\Ginclude@graphics{#2}}%
       \setkeys{Gin}{#1}%
       \Gin@esetsize
       \the\toks@
     \endgroup
     \fi}
```

5 Rotation

`\rotatebox` Look for an optional argument.

```
\def\rotatebox{%
  \@ifnextchar[\Grot@box@kv\Grot@box@std}
```

`\Grot@box@std` If no KV argument, just repeat the standard definition.

```
\def\Grot@box@std#1#2{%
  \Grot@setangle{#1}%
  \setbox\z@\hbox{{#2}}%
  \Grot@x\z@
  \Grot@y\z@
  \Grot@box}
```

\Grot@box@kv

```
\def\Grot@box@kv[#1]#2#3{%
  \@begin@tempboxa\hbox{#3}%
    \Grot@x\width \divide\Grot@x\tw@
    \Grot@y\height \advance\Grot@y-\depth \divide\Grot@y\tw@
    \setkeys{Grot}{#1}%
    \setbox\z@\box\@tempboxa
    \Grot@setangle{#2}%
    \Grot@box
  \@end@tempboxa}
```

There are two ways of specifying the centre of rotation.

\KV@Grot@origin `origin=`⟨*label*⟩, where the labels are up to two of `lrctbB` (`B` denotes the baseline, as for PSTricks).

```
\define@key{Grot}{origin}[c]{%
 \@tfor\@tempa:=#1\do{%
    \if l\@tempa \Grot@x\z@\else
    \if r\@tempa \Grot@x\width\else
    \if t\@tempa \Grot@y\height\else
    \if b\@tempa \Grot@y-\depth\else
    \if B\@tempa \Grot@y\z@\fi\fi\fi\fi\fi}}
```

\KV@Grot@x
\KV@Grot@y

`x=`⟨*dimen*⟩`,y=`⟨*dimen*⟩ The x, y coordinate of the centre of rotation. As usual `\height` etc may be used.

```
\define@key{Grot}{x}{\setlength\Grot@x{#1}}
\define@key{Grot}{y}{\setlength\Grot@y{#1}}
```

\KV@Grot@units 'units' specifies the number or units in one anti-clockwise circle. So the default is 360. −360 gives clockwise rotation, 6.283185 gives radians etc.

```
\define@key{Grot}{units}{%
  \def\Grot@setangle##1{%
  \dimen@##1\p@
  \dimen@ii#1\p@
  \divide\dimen@ii360\relax
  \divide\dimen@\dimen@ii
  \edef\Grot@angle{\number\dimen@}}}
```

\Gin@erotate Initialise the rotation command to use in `\includegraphics`.

```
\let\Gin@erotate\Grot@box@std
```

⟨/package⟩