

The `graphicx` package*

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

1999/02/16

1 Introduction

This package provides an alternative interface to the $\text{\LaTeX} 2_{\epsilon}$ 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 $\text{\LaTeX} 2_{\epsilon}$ 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 $\text{\LaTeX} 2.09$) 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 $\text{\LaTeX} 3$ 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 $\text{\LaTeX} 2_{\epsilon}$, which is intended as a ‘consolidation of existing \LaTeX variants’. The long term planning

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

for an eventual L^AT_EX3 release will then be able to consider the whole L^AT_EX 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}
```

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,bbly,bburx,bbury Set the bounding box. Mainly for compatibility with older packages. `bbllx=a,bbly=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,ly` 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 T_EX 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 referred 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 unknown `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@bblly
```

```
\KV@Gin@bblly
```

```
\KV@Gin@bburx
```

```
\KV@Gin@bbury
```

```
9 \define@key{Gin}{bblly}
10   {\Gin@bboxtrue\Gin@defaultbp\Gin@lly{#1}}
11 \define@key{Gin}{bblly}
12   {\Gin@bboxtrue\Gin@defaultbp\Gin@lly{#1}}
13 \define@key{Gin}{bburx}
14   {\Gin@bboxtrue\Gin@defaultbp\Gin@urx{#1}}
15 \define@key{Gin}{bbury}
16   {\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`.

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

```

\KV@Gin@natheight
\KV@Gin@natheight 22 \let\KV@Gin@natwidth\KV@Gin@bburx
23 \let\KV@Gin@natheight\KV@Gin@bbury

\KV@Gin@viewport A ‘viewport’ is a user-specified area of the graphic to be included. It should not
\KV@Gin@trim 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, im-
plemented) by Arthur Ogawa.
24 \define@key{Gin}{viewport}
25     {\let\Gin@viewport@code\Gin@viewport\Gread@parse@vp#1 \}
26 \define@key{Gin}{trim}
27     {\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 alterna-
tive macros that are used in the viewport and trim cases to modify the bounding
box read from the file.
28 \def\Gread@parse@vp#1 #2 #3 #4 #5\{
29 \Gin@defaultbp\Gin@vllx{#1}%
30 \Gin@defaultbp\Gin@villy{#2}%
31 \Gin@defaultbp\Gin@vurx{#3}%
32 \Gin@defaultbp\Gin@vury{#4}%

\Gin@viewport If a viewport is specified, reset the bounding box coordinates 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).
33 \def\Gin@viewport{%
34 \let\Gin@ollx\Gin@llx
35 \let\Gin@olly\Gin@lly
36 \let\Gin@ourx\Gin@urx
37 \let\Gin@oury\Gin@ury
38 \dimen@\Gin@llx\p@advance\dimen@ \Gin@vurx\p@
39     \edef\Gin@urx{\strip@pt\dimen@}%
40 \dimen@\Gin@lly\p@advance\dimen@ \Gin@vury\p@
41     \edef\Gin@ury{\strip@pt\dimen@}%
42 \dimen@\Gin@llx\p@advance\dimen@ \Gin@vllx\p@
43     \edef\Gin@llx{\strip@pt\dimen@}%
44 \dimen@\Gin@lly\p@advance\dimen@ \Gin@villy\p@
45     \edef\Gin@lly{\strip@pt\dimen@}}

\Gin@trim If a trim is specified, reset the bounding box coordinates by trimming the four
specified values off each side of the graphic.
46 \def\Gin@trim{%
47 \let\Gin@ollx\Gin@llx
48 \let\Gin@olly\Gin@lly

```

```

49 \let\Gin@ourx\Gin@urx
50 \let\Gin@oury\Gin@ury
51 \dimen@\Gin@llx\p@\advance\dimen@ \Gin@vllx\p@
52 \edef\Gin@llx{\strip@pt\dimen@}%
53 \dimen@\Gin@lly\p@\advance\dimen@ \Gin@vly\p@
54 \edef\Gin@lly{\strip@pt\dimen@}%
55 \dimen@\Gin@urx\p@\advance\dimen@ -\Gin@vurx\p@
56 \edef\Gin@urx{\strip@pt\dimen@}%
57 \dimen@\Gin@ury\p@\advance\dimen@ -\Gin@vury\p@
58 \edef\Gin@ury{\strip@pt\dimen@}}

```

`\Gin@vllx` `\Gin@vly` Four macros to hold the modifiers for the bounding box for viewport and trim specifications.

`\Gin@vurx` `\Gin@vury` 59 `\let\Gin@vllx\Gin@llx\let\Gin@vly\Gin@lly`
60 `\let\Gin@vurx\Gin@urx\let\Gin@vury\Gin@ury`

`\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.

```

61 \define@key{Gin}{angle}
62     {\Gin@esetsize
63     \@tempwattrue
64     \edef\@tempa{\toks@{\noexpand\Gin@erotate{#1}{\the\toks@}}}%
65     \@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.

```

66 \define@key{Gin}{origin}[c]{%
67 \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.

```

68 \define@key{Gin}{width}{\def\Gin@ewidth{#1}}
69 \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`.

```

70 \define@key{Gin}{totalheight}{%
71 \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.

```

72 \define@key{Gin}{keepaspectratio}[true]{%
73 \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.

```

74 \define@key{Gin}{scale}{%
75   \if@tempswa
76     \edef\@tempa{\toks@{\noexpand\Gscale@box{#1}[#1]{\the\toks@}}}%
77     \@tempa
78   \else
79     \def\Gin@req@sizes{%
80       \def\Gin@scalex{#1}\let\Gin@scaley\Gin@exclamation
81       \Gin@req@height\Gin@scalex\Gin@nat@height
82       \Gin@req@width\Gin@scalex\Gin@nat@width}%
83   \fi
84   \@tempwatruel}

```

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

```

85 \define@key{Gin}{draft}[true]{%
86   \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.

```

87 \define@key{Gin}{clip}[true]{%
88   \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’.

```

89 \define@key{Gin}{type}{%
90   \def\Gin@include@graphics##1{%
91     \begingroup
92     \def\Gin@base{##1}%
93     \edef\@tempa{##1}{\Gin@eread}{\Gin@ecom{##1\Gin@eext}}}%
94     \expandafter\Gin@setfile\@tempa
95     \endgroup}}

```

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

```

96 \define@key{Gin}{ext}{\def\Gin@eext{#1}}
97 \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`.

```

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

```

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

```

102 \define@key{Gin}{command}{\def\Gin@ecom##1{#1}}
103 \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.

```

104 \def\Gin@boolkey#1#2{%
105   \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.

```

106 \def\Gin@eresize{\height}
107 \def\Gin@esetsize{%
108   \let\@tempa\Gin@exclamation
109   \if@tempswa

```

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

```

110   \edef\@tempa{\toks@{\noexpand
111     \scale@box\noexpand\Gin@eresize
112     {\Gin@ewidth}{\Gin@eheight}{\the\toks@}}}%
113   \@tempa
114   \else

```

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

```

115   \ifx\Gin@ewidth\@tempa
116     \ifx\Gin@eheight\@tempa

```

No resizing.

```

117     \else

```

Just height specified.

```

118       \let\Gin@@eheight\Gin@eheight
119       \def\Gin@req@sizes{%
120         \Gscale@div\Gin@scaley\Gin@@eheight\Gin@nat@height
121         \let\Gin@scalex\Gin@exclamation
122         \setlength\Gin@req@height\Gin@@eheight
123         \Gin@req@width\Gin@scaley\Gin@nat@width}%
124       \fi
125     \else
126       \ifx\Gin@eheight\@tempa

```

Just width specified.

```

127       \let\Gin@@ewidth\Gin@ewidth
128       \def\Gin@req@sizes{%
129         \Gscale@div\Gin@scalex\Gin@@ewidth\Gin@nat@width
130         \let\Gin@scaley\Gin@exclamation
131         \setlength\Gin@req@width\Gin@@ewidth
132         \Gin@req@height\Gin@scalex\Gin@nat@height}%
133     \else

```

Both height and width specified.

```
134     \let\Gin@@ewidth\Gin@ewidth
135     \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!.

```
136     \def\Gin@nosize##1{%
137         \KV@Gin@natwidth\Gin@@ewidth
138         \KV@Gin@natheight\Gin@@eheight}%
139     \def\Gin@req@sizes{%
140         \Gscale@div\Gin@scalex\Gin@@ewidth\Gin@nat@width
141         \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.

```
142     \ifGin@iso
143         \ifdim\Gin@scaley\p@>\Gin@scalex\p@
144             \let\Gin@scaley\Gin@scalex
145         \else
146             \let\Gin@scalex\Gin@scaley
147         \fi
148     \fi
149     \Gin@req@width\Gin@scalex\Gin@nat@width
150     \Gin@req@height\Gin@scaley\Gin@nat@height}%
151 \fi
152 \fi
153 \fi
154 \let\Gin@ewidth\Gin@exclamation
155 \let\Gin@eheight\Gin@ewidth}
```

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

```
\Gin@req@width 156 \newdimen\Gin@req@height
157 \newdimen\Gin@req@width
```

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

```
\Gin@outer@scaley 158 \let\Gin@outer@scalex\relax
159 \let\Gin@outer@scaley\relax
```

`\Gin@angle` Rotation angle.

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

```

\Gin@ewidth Final size, initialised for no scaling.
\Gin@eheight 161 \let\Gin@ewidth\Gin@exclamation
162 \let\Gin@eheight\Gin@ewidth

\Gin@scalex Scale factors. Initialised for no scaling.
\Gin@scaley 163 \def\Gin@scalex{1}
164 \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.
165 \def\Gin@i{%
166 \def\Gin@req@sizes{%
167 \Gin@req@height\Gin@nat@height
168 \Gin@req@width\Gin@nat@width}%
169 \@ifnextchar[\Gin@ii{\Gin@ii []}]

\Gin@ii Look for a second optional argument. If one optional argument is present, call
\setkeys to process it,
170 \def\Gin@ii[#1]#2{%
171 \def\@tempa{[]}\def\@tempb{#2}%
172 \ifx\@tempa\@tempb
173 \def\@tempa{\Gin@iii[#1] []}%
174 \expandafter\@tempa
175 \else
176 \begingroup
177 \@tempswafalse
178 \toks@{\Gin@include@graphics{#2}}%
179 \setkeys{Gin}{#1}%
180 \Gin@esetsize
181 \the\toks@
182 \endgroup
183 \fi}

```

5 Rotation

```

\rotatebox Look for an optional argument.
184 \def\rotatebox{%
185 \@ifnextchar[\Grot@box@kv\Grot@box@std}

\Grot@box@std If no KV argument, just repeat the standard definition.
186 \def\Grot@box@std#1#2{%
187 \Grot@setangle{#1}%
188 \setbox\z@\hbox{#2}%
189 \Grot@x\z@
190 \Grot@y\z@
191 \Grot@box}

```

```

\Grot@box@kv
192 \def\Grot@box@kv[#1]#2#3{%
193   \@begin@tempboxa\hbox{#3}%
194     \Grot@x\width \divide\Grot@x\tw@
195     \Grot@y\height \advance\Grot@y-\depth \divide\Grot@y\tw@
196     \setkeys{Grot}{#1}%
197     \setbox\z@\box\@tempboxa
198     \Grot@setangle{#2}%
199     \Grot@box
200   \@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`).

```

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

```

`\KV@Grot@x` `x=<dimen>`, `y=<dimen>` The x, y coordinate of the centre of rotation. As usual `\KV@Grot@y` `\height` etc may be used.

```

208 \define@key{Grot}{x}{\setlength\Grot@x{#1}}
209 \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.

```

210 \define@key{Grot}{units}{%
211   \def\Grot@setangle##1{%
212     \dimen@##1\p@
213     \dimen@ii#1\p@
214     \divide\dimen@ii360\relax
215     \divide\dimen@\dimen@ii
216     \edef\Grot@angle{\number\dimen@}}

```

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

```

217 \let\Gin@erotate\Grot@box@std
218 \end{package}

```