Packages in the 'graphics' bundle

D. P. Carlisle

1995/12/07

Contents

1 Introduction

This document serves as a user-manual for the packages color, graphics, and graphicx. Further documentation may be obtained by processing the source (dtx) files of the individual packages.

2 Driver support

All these packages rely on features that are not in TEX itself. These features must be supplied by the 'driver' used to print the `dvi` file. Unfortunately not all drivers support the same features, and even the internal method of accessing these extensions varies between drivers. Consequently all these packages take options such as '`dvips`' to specify which driver is being used.

3 Colour

The colour support is built around the idea of a system of *Colour Models*. The Colour models supported by a driver vary, but typically include

rgb Red Green Blue: A comma separated list of three numbers between 0 and 1, giving the components of the colour.

cmyk Cyan Magenta Yellow [K]Black: A comma separated list of four numbers between 0 and 1, giving the components of the colour according to the additive model used in most printers.

gray Grey scale: a single number between 0 and 1.

named Colours accessed by name, e.g. 'JungleGreen'. Not all drivers support this model. The names must either be 'known' to the driver or added using commands described in `color.dtx`. Some drivers support an extended form of the named model in which an 'intensity' of the colour may also be specified, so 'JungleGreen, 0.5' would denote that colour at half strength.

Note that the **named** model is really just given as an example of a colour model that takes names rather than a numeric specification. Other options may be provided locally that provide different colour models, eg **pantone** (An industry standard set of colours), **x11** (Colour names from the X Window System), etc. The standard distribution does not currently have such models, but the **named** model could be used as an example of how to define a new colour model. The names used in the **named** model are those suggested by Jim Hafner in his colordvi and foiltex packages, and implemented originally in the `color.pro` header file for the `dvips` driver.

3.1 Package Options

Most of the options to the color package just specify a driver, e.g. `dvips`. You should to set up a site default for these options, for the driver that you normally use. Suppose that you wish for the color package to always default to use specials for the PostScript driver, `dvipsone`. In that case create a file `color.cfg` containing the line:
`\ExecuteOptions{dvipsone}`

One special option that is of interest is `monochrome`. If this option is selected the colour commands are all disabled so that they do not generate errors, but do not generate colour either. This is useful if previewing with a previewer that can not produce colour.

Three other package options control the use of the **named** model. The `dvips` driver (by default) pre-defines 68 colour names. The `dvips` option normally makes these names available in the **named** colour model. If you do not want these names to be declared in this model (Saving TEX some memory) you may give the `nodvipsnames` option. Conversely, if you are using another driver, you may wish to add these names to the named model for that driver (especially if you are processing a document originally produced on `dvips`). In this case you could use the `dvipsnames` option. Lastly the `usenames` option makes all names in the **named** model directly available, as described below.

3.2 Defining Colours

The colours `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow` should be predefined, but should you wish to mix your own colours use the `\definecolor` command.

`\definecolor{`*name*`}{`*model*`}{`*colour specification*`}`

This defines ⟨*name*⟩ as a colour which can be used in later colour commands. For example

```
\definecolor{light-blue}{rgb}{0.8,0.85,1}
\definecolor{mygrey}{gray}{0.75}
```

Now `light-blue` and `mygrey` may be used in addition to the predefined colours above.

3.3 Using Colours

3.3.1 Using predefined colours

The syntax for colour changes is designed to mimic font changes. The basic syntax is:

`\color{`*name*`}`

This is a *declaration*, like `\bfseries` It changes the current colour to ⟨*name*⟩ until the end of the current group or environment.

An alternative command syntax is to use a *command* form that takes the text to be coloured as an *argument*. This is similar to the font commands such as `\textbf`:

`\textcolor{`*name*`}{`*text*`}`

So the above is essentially equivalent to `{\color{`*name*`}`*text*`}`.

3.3.2 Using colour specifications directly

```
\color[⟨model⟩]{specification}
\textcolor[⟨model⟩]{specification}{text}
```

Normally one would predeclare all the colours used in a package, or in the document preamble, but sometimes it is convenient to directly use a colour without naming it first. To achieve this `\color` (and all the other colour commands) take an optional argument specifying the model. If this is used then the mandatory argument takes a ⟨*colour specification*⟩ instead of a ⟨*name*⟩. For example:
`\color[rgb]{1,0.2,0.3}`
would directly select that colour.

This is particularly useful for accessing the **named** model:
`\color[named]{BrickRed}` selects the `dvips` colour BrickRed.

Rather than repeatedly use `[named]` you may use `\definecolor` to provide convenient aliases:
`\definecolor{myred}{named}{WildStrawberry} ... \color{myred} ...`

Alternatively if you are happy to use the existing names from the **named** model, you may use the `usenames` package option, which effectively calls `\definecolor` on every colour in the **named** model, thus allowing `\color{WildStrawberry}` in addition to `\color[named]{WildStrawbery}`.

3.4 Named Colours

Using the **named** colour model has certain advantages over using other colour models.

Firstly as the `dvi` file contains a request for a colour by *name*, the actual mix of primary colours used to obtain the requested colour can be tuned to the characteristics of a particular printer. In the `dvips` driver the meanings of the colour names are defined in the header file `color.pro`. Users are encouraged to produce different versions of this file for any printers they use. By this means the same dvi file should produce colours of similar appearance when printed on printers with different colour characteristics.

Secondly, apart from the so called 'process colours' that are produced by mixing primary colours during the print process, one may want to use 'spot' or 'custom' colours. Here a particular colour name does not refer to a mix of primaries, but to a particular ink. The parts of the document using this colour will be printed separately using this named ink colour.

3.5 Page Colour

```
\pagecolor{name}
```

The background colour of the whole page can be set using `\pagecolor`. This takes the same argument forms as `\color` but sets the background colour for the current and all subsequent pages. It is a global declaration, so you need to use `\pagecolor{white}` to 'get back to normal'.

3.6 Box Backgrounds

Two commands similar to \fbox produce boxes with the backgrounds shaded an appropriate colour.

```
\colorbox{name}{text}
\fcolorbox{name1}{name2}{text}
```

The former produces a box coloured with *name* like this. The latter is similar but puts a frame of colour *name1* around the box coloured *name2*.

These commands use the \fbox parameters \fboxrule and \fboxsep to determine the thickness of the rule, and the size of the shaded area.

3.7 Possible Problems

TeX was not designed with colour in mind, and producing colours requires a lot of help from the driver program. Thus, depending on the driver, some or all features of the **color** package may not be available.

Some drivers do not maintain a special 'colour stack'. These drivers are likely to get confused if you nest colour changes, or use colours in floating environments.

Some drivers do not maintain colours over a page break, so that if the page breaks in the middle of a coloured paragraph, the last part of the text will incorrectly be printed in black.

There is a different type of problem that will occur for all drivers. Due to certain technical difficulties[1], it is possible that at points where the colour changes, the *spacing* is affected. For this reason the monochrome option does not completely disable the colour commands, it redefines them to write to the log file. This will have the same effects on spacing, so you can produce monochrome drafts of your document, at least knowing that the final spacing is being shown.

4 The Graphics packages

There are two graphics packages:

graphics The 'standard' graphics package.

graphicx The 'extended' or 'enhanced' graphics package.

The two differ only in the format of optional arguments for the commands defined. The command names, and the mandatory arguments are the same for the two packages.

[1] At least two causes: 1) The presence of a \special ⟨*whatsit*⟩ prevents \addvspace 'seeing' space on the current vertical list, so causing it to incorrectly add extra vertical space. 2) A ⟨*whatsit*⟩ as the first item in a \vtop moves the reference point of the box.

4.1 Package Options

The graphics packages share the same 'driver' options as the `color` package. As for colour you should set up a site-default in a file, `graphics.cfg`, containing the line (for `dvips`):
`\ExecuteOptions{dvips}`

The graphics packages have some other options for controlling how many of the features to enable:

draft suppress all the 'special' features. In particular graphics files are not included (but they are still read for size info) just the filename is printed in a box of the correct size.

final The opposite of `draft`. Useful to over-ride a global `draft` option specified in the `\documentclass` command.

hiderotate Do not show rotated text (presumably because the previewer can not rotate).

hidescale Do not show scaled text (presumably because the previewer can not scale).

4.2 Rotation

graphics:	`\rotatebox{`*angle*`}{`*text*`}`
graphicx:	`\rotatebox[`⟨*key val list*⟩`]{`*angle*`}{`*text*`}`

This puts *text* in a box, like `\mbox`, but rotates the box through *angle* degrees, like this.

The standard version always rotates around the reference point of the box, but the keyval version takes the following keys:

`origin=`⟨*label*⟩
`x=`⟨*dimen*⟩
`y=`⟨*dimen*⟩
`units=`⟨*number*⟩

So you may specify both `x` and `y`, which give the coordinate of the centre of rotation relative to the reference point of the box, eg `[x=2mm, y=5mm]`. Alternatively, for the most common points, one may use `origin` with a *label* containing one or two of the following: `lrctbB` (`B` denotes the baseline, as for PSTricks). For example, compare a default rotation of 180° ... Like This ... to the effects gained by using the `origin` key:
`[origin = c]` rotates about the centre of the box,... Like This...
`[origin = tr]` rotates about the top right hand corner... Like This ...

The `units` key allows a change from the default units of degrees anti-clockwise. Give the number of units in one full anti-clockwise rotation. For example:
`[units = -360]` specifies degrees clockwise.
`[units= 6.283185]` specifies radians.

4.3 Scaling

4.3.1 Scaling by scale factor

`\scalebox{`*h-scale*`}[`⟨*v-scale*⟩`]{`*text*`}`

Again this is basically like `\mbox` but scales the *text*. If *v-scale* is not specified it defaults to *h-scale*. If it is specified the text is distorted as the horizontal and vertical stretches are different, Like This.

`\reflectbox{`*text*`}`

An abbreviation for `\scalebox{-1}[1]{`*text*`}`.

4.3.2 Scaling to a requested size

`\resizebox*{`*h-length*`}{`*v-length*`}{`*text*`}`

Scale *text* so that the width is *h-length*. If `!` is used as either length argument, the other argument is used to determine a scale factor that is used in both directions. Normally *v-length* refers to the height of the box, but in the star form, it refers to the 'height + depth'. As normal for LaTeX 2ε box length arguments, `\height`, `\width`, `\totalheight`, `\depth` may be used to refer to the original size of the box.

`\resizebox{1in}{\height}{Some text}`: Some text

`\resizebox{1in}{!}{Some text}`: Some text

4.4 Including Graphics Files

The functions for graphics inclusion try to give the same user syntax for including any kind of graphics file that can be understood by the driver. This relies on the file having an extension that identifies the file type. The 'driver options' will define a collection of file extensions that the driver can handle, although this list may be extended using the declarations described below.

If the file's extension is unknown to the driver, the system may try a default file type. The PostScript driver files set this default to be eps (PostScript), but this behaviour may be customised if other defaults are required.

graphics: `\includegraphics*[`⟨*llx,lly*⟩`][`⟨*urx,ury*⟩`]{`*file*`}`
graphicx: `\includegraphics*[`⟨*key val list*⟩`]{`*file*`}`

Include a graphics file.

If `*` is present, then the graphic is 'clipped' to the size specified. If `*` is omitted, then any part of the graphic that is outside the specified 'bounding box' will over-print the surrounding text.

If the optional arguments are omitted, then the size of the graphic will be determined by reading an external file as described below.

graphics version If `[`⟨*urx,ury*⟩`]` is present, then it should specify the coordinates of the top right corner of the image, as a pair of TeX dimensions. If the units are omitted they default to `bp`. So `[1in,1in]` and `[72,72]` are equivalent. If only one optional argument appears, the lower left corner of the image is assumed to be at `[0,0]`. Otherwise `[`⟨*llx,lly*⟩`]` may be used to specify the coordinates of this point.

graphicx version Here the star form is just for compatibility with the standard version. It just adds `clip` to the list of keys specified. (Also, for increased compatibility, if *two* optional arguments are used, the 'standard' version of `\includegraphics` is always used, even if the graphicx package is loaded.)

The allowed keys are listed below.

bb The argument should be four dimensions, separated by spaces. These denote the 'Bounding Box' of the printed region within the file.

bbllx,bblly,bburx,bbury Set the bounding box. Mainly for compatibility with older packages. Specifying `bbllx=a,bblly=b,bburx=c,bbury=d` is equivalent to specifying `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 The `viewport` key takes four arguments, just like `bb`. However in this case the values are taken relative to the origin specified by the bounding box in the file. So to 'view' the 1in square in the bottom left hand corner of the area specified by the bounding box, use the argument `viewport=0 0 72 72`. New feature 1995/06/01

trim Similar to viewport, but here the four lengths specify the amount to remove or add to each side. `trim= 1 2 3 4` 'crops' the picture by 1bp at the left, 2bp at the bottom, 3bp on the right and 4bp at the top. New feature 1995/06/01

angle Rotation angle.

origin Origin for rotation. See the documentation of `\rotatebox`. New feature 1995/09/28

width Required width. The graphic is scaled to this width.

height Required height. The graphic is scaled to this height.

totalheight Specify the total height (height + depth) of the figure. This will differ from the 'height' if rotation has occurred. In particular if the figure has been rotated by $-90°$ then it will have zero height but large depth. New feature 1995/06/01

keepaspectratio Boolean valued key like 'clip'. If set to true then specifying both 'width' and 'height' (or 'totalheight') does not distort the figure but scales such that neither of the specified dimensions is *exceeded*. New feature 1995/09/27

scale Scale factor.

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

draft a boolean valued key, like ‘clip’. Locally switches to draft mode.

type Specify the graphics type.

ext Specify the file extension.

read Specify the file extension of the ‘read file’.

command Specify any command to be applied to the file.

For the keys specifying the original size (ie the bounding box, trim and viewport keys) the units can be omitted, in which case bp (ie PostScript points) are assumed.

The first seven keys specify the original size of the image. This size needs to be specified in the case that the file can not be read by TEX, or it contains an incorrect size ‘BoundingBox’ specification.

`bbllx`... `\bbury` are mainly for compatibility for older packages.
`bbllx=a, bblly=b, bburx=c, bbury=d`
is equivalent to
`bb = a b c d`.

`natheight` and `natwidth` are just shorthands for setting the lower left coordinate to 0 0 and the upper right coordinate to the specified width and height.

The next few keys specify any scaling or rotation to be applied to the image. To get these effects using the standard package, the `\includegraphics` call must be placed inside the argument of a `\rotatebox` or `\scalebox` command.

The keys are read left-to-right, so `[angle=90, height=1in]` means rotate by 90 degrees, and then scale to a height of 1in. `[height=1in, angle=90]` would result in a final *width* of 1in.

If the calc package is also loaded the lengths may use calc syntax, for instance to specify a width of 2 cm less than the text width: `[width=\textwidth-2cm]`.

TEX leaves the space specified either in the file, or in the optional arguments. If any part of the image is actually outside this area, it will by default overprint the surrounding text. If the star form is used, or `clip` specified, any part of the image outside this area will not be printed.

The last four keys suppress the parsing of the filename. If they are used, the main *file* argument should not have the file extension. They correspond to the arguments of `\DeclareGraphicsRule` described below.

To see the effect that the various options have consider the file `a.ps`. This file contains the bounding box specification

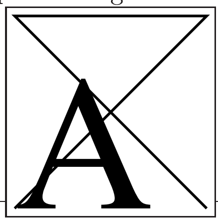
```
%%BoundingBox:100 100 172 172
```

That is, the printed region consists of a one-inch square, 100 pt in from the bottom and left hand edges of the paper.

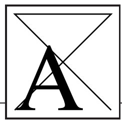
In all the following examples the input will be of the form `left---\fbox{\includegraphics{a}}---right`

With different options supplied to `\includegraphics`.

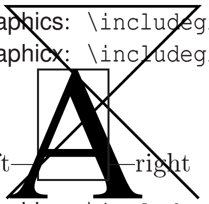
No optional argument.

left—  —right

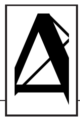
graphics: `\scalebox{0.5}{\includegraphics{a}}`
graphicx: `\includegraphics[scale=.5]{a}`

left—  —right

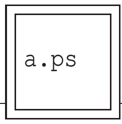
graphics: `\includegraphics[115,110][135,145]{a}}`
graphicx: `\includegraphics[bb= 115 110 135 145]{a}`

left—  —right

graphics: `\includegraphics*[115,110][135,145]{a}}`
graphicx: `\includegraphics[bb= 115 110 135 145,clip]{a}`

left—  —right

graphics: `\scalebox{0.5}{\includegraphics{a}}` and `draft` option.
graphicx: `\includegraphics[scale=.5, draft]{a}`

left— 

—right

4.5 Other commands in the graphics package

`\graphicspath{`*dir-list*`}`

This optional declaration may be used to specify a list of directories in which to search for graphics files. The format is the same as for the LaTeX 2ε primitive `\input@path`. A list of directories, each in a `{}` group (even if there is only one in the list). For example:
`\graphicspath{{eps/}{tiff/}}`
would cause the system to look in the subdirectories `eps` and `tiff` of the current directory. The default setting of this path is `\input@path` that is: graphics files will be found wherever TeX files are found.

`\DeclareGraphicsExtensions{`*ext-list*`}`

This specifies the behaviour of the system when no file extension is specified in the argument to `\includegraphics`. `{`*ext-list*`}` should be a comma separated list of file extensions. (White space is ignored between the entries.) A file name is produced by appending one extension from the list. If a file is found, the system acts as if that extension had been specified. If not, the next extension in *ext-list* is tried. New description 1994/12/01

Note that if the extension is not specified in the `\includegraphics` command, the graphics file must exist at the time LaTeX is run, as the existence of the file is used to determine which extension from the list to choose. However if a file extension *is* specified, e.g. `\includegraphics{a.ps}` instead of `\includegraphics{a}`, then the graphics file need not exist at the time LaTeX is used. (In particular it may be created on the fly by the ⟨*command*⟩ specified in the `\DeclareGraphicsRule` command described below.) LaTeX does however need to be able to determine the size of the image so this size must be specified in arguments, or the 'read file' must exist at the time LaTeX is used.

`\DeclareGraphicsRule{`*ext*`}{`*type*`}{`*read-file*`}{`*command*`}`

Any number of these declarations can be made. They determine how the system behaves when a file with extension *ext* is specified. (The extension may be specified explicitly or, if the argument to `\includegraphics` does not have an extension, it may be a default extension from the *ext-list* specified with `\DeclareGraphicsExtensions`.)

ext the file extension for which this rule applies. As a special case, *ext* may be given as `*` to denote the default behaviour for all undeclared extensions (see the example below).

type is the 'type' of file involved. All files of the same type will be input with the same internal command (which must be defined in a 'driver file'). For example files with extensions `ps`, `eps`, `ps.gz` may all be classed as type `eps`.

read-file determines the extension of the file that should be read to determine size information. It may be the same as *ext* but it may be different, for example `.ps.gz` files are not readable easily by TeX, so you may want to put the bounding box information in a separate file with extension `.ps.bb`. If *read-file* is empty, `{}`, then the system will not try to locate an external file for size info, and the size must be specified in the arguments of `\includegraphics`. If the driver file specifies a procedure for reading size files for *type*, that will be used, otherwise the procedure for reading `eps` files will be used. Thus the size of bitmap files may be specified in a file with a PostScript style `%%BoundingBox` line, if no other specific format is available.

As a special case `*` may be used to denote the same extension as the graphic file. This is mainly of use in conjunction with using `*` as the extension, as in that case the particular graphic extension is not known. For example

```
\DeclareGraphicsRule{*}{eps}{*}{}
```

This would declare a default rule, such that all unknown extensions would be treated as EPS files, and the graphic file would be read for a BoundingBox comment.

command is usually empty, but if non empty it is used in place of the filename in the `\special`. Within this argument, `#1` may be used to denote the filename. Thus using the dvips driver, one may use
`\DeclareGraphicsRule{.ps.gz}{eps}{.ps.bb}{`zcat #1}`
the final argument causes dvips to use the `zcat` command to unzip the file before inserting it into the PostScript output.

4.6 Global setting of keys

Most of the keyval keys used in the graphicx package may also be set using the command `\setkeys` provided by the keyval package.

For instance, suppose you wanted all the files to be included in the current document to be scaled to 75% of the width of the lines of text, then one could issue the following command:
`\setkeys{Gin}{width=0.75\textwidth}`
Here '`Gin`' is the name used for the keyval keys associated with 'Graphics inclusion'. All following `\includegraphics` commands (within the same group or environment) will act as if `[width=0.75\textwidth]` had been specified, in addition to any other key settings actually given in the optional argument.

Similarly to make all `\rotatebox` arguments take an argument in radians, one just needs to specify:
`\setkeys{Grot}{units=6.28318}`

4.7 Compatibility between graphics and graphicx

For a document author, there are not really any problems of compatibility between the two packages. You just choose the interface that you personally prefer, and then use the appropriate package.

For a package or class writer the situation is slightly different. Suppose that you are writing a letter class that needs to print a company logo as part of the letterhead.

As the author of the class you may want to give the users the possibility of using either interface in their letters (should they need to include any further graphics into the letter body). In this case the class should load the graphics package (not graphicx, as this would commit any users of the class to the keyval interface). The logo should be included with `\includegraphics` either with *no* optional argument (if the correct size information is in the file) or *both* optional arguments otherwise. Do not use the *one* optional argument form, as the meaning of this argument would change (and generate errors) if the user were to load graphicx as well as your class.

5 Remaining packages in the graphics bundle

5.1 Epsfig

This is a small package essentially a ‘wrapper’ around the graphicx package, defining a command `\psfig` which has the syntax
`\psfig{file=xxx,...}` rather than `\includegraphics[...]{xxx}`.
It also has a few more commands to make it slightly more compatible with the old LaTeX 2.09 style of the same name.

5.2 Trig

The trig package is not intended to be used directly in documents. It calculates sine, cosine and tangent trigonometric functions. These are used to calculate the space taken up by a rotated box. This package is also used by the `fontinst` program which converts PostScript files to a form usable by TeX.

As well as being used as a LaTeX package, the macros may be extracted with the docstrip options `plain,package`. In this case the LaTeX package declarations are omitted from the file, and the macros may be directly used as part of another macro file (they work with any format based on plain TeX.)

5.3 Keyval

The keyval package is intended to be used by other packages. It provides a generic way of setting ‘keys’ as used by the graphicx package, and splitting up the comma separated lists of $\langle key\rangle = \langle value\rangle$ pairs.

Like the trig package, these macros may be extracted and used as part of another macro file, based on plain TeX, as well as the standard use as a LaTeX package.

5.4 Pstcol

PSTricks, by Timothy Van Zandt is an immensely powerful package that enables a very full featured interface between PostScript and TeX. Unfortunately the colour support in PSTricks is slightly incompatible with the colour mechanism defined in the color package. The pstcol package is a (hopefully temporary) package that modifies a very small number of internal PSTricks functions, to remove this incompatibility. If pstricks is loaded via this package, you may use any colours defined by color package commands within pstricks commands, and vice versa.

5.5 Lscape

The lscape package requires and takes the same options as the graphics package. It defines a `landscape` environment within which page bodies are rotated through 90 degrees. The page head and foot are not affected, they appear in the standard (portrait) position.