The graphics package*

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

2009/02/05

1 Introduction

This package implements various 'graphics' functions. The main features are a) inclusion of 'graphics' files. b) Rotation of sections of the page, c) Scaling of sections of the page.

The design is split into three 'levels'.

- The user interface. This is the collection of commands designed to appear in a document text. Actually two separate user interface have been implemented. The 'standard' interface, described here, and a more powerful, and more 'user-friendly' interface provided by the `graphicx` package.
- The core functions. These functions, which are also implemented in this file do all the 'main work'. The 'user-interface functions just collect together the information from any optional-arguments or star-forms, and then call one of these functions.
- The driver files. It is not possible to achieve the functionality of this package just using TEX. The `dvi` driver used must be given additional instructions. (Using the `\special` command of TEX.) Unfortunately, the capabilities of various drivers differ, and the syntax required to pass instructions to the drivers is also not standardised. So the 'core functions' never access `\special` directly, but rather call a series of commands that must be defined in a special file customised for each driver. The accompanying file, `drivers.dtx` has suitable files for a range of popular drivers.

2 Package Options

Most of the options, such as `dvips`, `textures` etc., specify the driver that is to be used to print the document. You may wish to set up a configuration file so that this option always takes effect, even if not specified in the document. To do this, produce a file `graphics.cfg` containing the line:
`\ExecuteOptions{dvips}`
(or whichever other driver you wish.)

Apart from the driver options there are a few other options to control the behaviour of the package.

draft Do not include graphics files, but instead print a box of the size the graphic would take up, and the file name. This greatly speeds up previewing on most systems.

final Turns off the `draft` option.

debugshow Show a lot of tracing information on the terminal. If you are not me you probably do not want to use this option.

*This file has version number v1.0o, last revised 2009/02/05.

hiderotate Do not show rotated text. Sometimes useful if your previewer can not rotate text.

hidescale Do not show scaled text.

hiresbb Look for Bounding Box lines of the form `%%HiResBoundingBox` instead of the standard `%%BoundingBox`. These are used by some applications to get round the restriction that BoundingBox comments should only have integer values.

demo Instead of including a graphics file, make `\includegraphics` insert a black rectangle of size 150 pt by 100 pt unless either dimension was already specified by another option.

3 Standard Interface

3.1 Graphics Inclusion

`\includegraphics` `*[`⟨*llx,lly*⟩`][`⟨*urx,ury*⟩`]{`⟨*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 overprint 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. 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 ommited 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 cordinates of this point.

`\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 as for the LATEX 2_ε 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 the argument to `\includegraphics` does not have an extension specified. Here `{`⟨*ext-list*⟩`}` should be a comma-separated list of file extensions, each with a leading period (`.`). A file name is produced by appending *sep* and one extension. 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.

Each use of `\DeclareGraphicsExtensions` overwrites all previous definitions. It is not possible to add an extension to an existing list.

Early versions of this package defined a default argument for this command. This has been removed.

`\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 is the *extension* of the file. Any file with this extension will be processed by this graphics rule. Normally a file with an extension for which no rule has been declared will generate an error, however you may use `*` as the extension to define

a *default rule*. For instance the `dvips` driver file declares all files to be of type `eps` unless a more specific rule is declared.

Since Version v0.6, extensions should be specified including the `.` that is, `.eps` not `eps`.

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`. As a special case `*` may be used to denote the same extension as the graphic file. This is mainly of use in conjuction 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.

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.

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.

3.2 Rotation

`\rotatebox` `{`⟨*angle*⟩`}{`⟨*text*⟩`}`
Rotate *text* *angle* degrees anti-clockwise. Normally the rotation is about the left-hand end of the baseline of *text*.

3.3 Scaling

`\scalebox` `{`⟨*h-scale*⟩`}[`⟨*v-scale*⟩`]{`⟨*text*⟩`}`
Scale *text* by the specified amounts. If *v-scale* is omitted, the vertical scale factor is the same as the horizontal one.

`\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 2_ε box length arguments, `\height`, `\width`, `\totalheight` and `\depth` may be used to refer to the original size of the box.

4 The Key=Value Interface

As mentioned in the introduction, apart from the above 'standard interface', there is an alternative syntax to the `\includegraphics` and `\rotatebox` commands that some people may prefer. It is provided by the accompanying `graphicx` package.

5 The Graphics Kernel Functions

5.1 Graphics Inclusion

`\Ginclude@graphics` `{`⟨*file*⟩`}`
Insert the contents of the file *file* at the current point. `\Ginclude@graphics` may use the four macros `\Gin@llx`, `\Gin@lly`, `\Gin@urx`, `\Gin@ury` to determine the 'bounding box' of the graphic. The result will be a TEX box of width $urx - llx$ and height $ury - lly$. If `\Gin@clip` is ⟨*true*⟩ then part of the graphic that is outside this box should not be displayed. (Not all drivers can support this 'clipping'.) Normally all these parameters are set by the 'user interface level'.

`\Gread@eps` `{`⟨*file*⟩`}`
For each *type* of graphics file supported, the driver file must define `\Ginclude@`*type* and, optionally `\Gread@`*type*. The read command is responsible for obtaining size information from the file specified in the `\DeclareGraphicsRule` command. However the kernel defines a function, `\Gread@eps`, which can read PostScript files to find the `%%BoundingBox` comment. This function will be used for any type for which a specific function has not been declared. `\Gread@eps` accepts a generalised version of the bounding box comment. TEX units may be used (but there must be no space before the unit). If the unit is omitted `bp` is assumed. So
`%%BoundingBox 0 0 2in 3in`
Would be accepted by this function, to produce a 2in wide, by 3in high graphic.

5.2 Rotation

`\Grot@box`
Rotate the contents of `\box0` through `\Grot@angle` degrees (anti-clockwise). The user-interface is responsible for setting the macro `\Grot@angle`, and putting the appropriate text in `\Grot@box`.

5.3 Scaling

`\Gscale@box` `{`⟨*xscale*⟩`}[`⟨*yscale*⟩`]{`⟨*text*⟩`}`
(The second argument is not optional.) Scale *text* by the appropriate scale factors.

`\Gscale@box@dd` `{`⟨*dima*⟩`}{`⟨*dimb*⟩`}{`⟨*text*⟩`}`
Scale *text* in both directions by a factor *dima*/*dimb*.

`\Gscale@box@dddd` `{`⟨*dima*⟩`}{`⟨*dimb*⟩`}{`⟨*dimc*⟩`}{`⟨*dimd*⟩`}{`⟨*text*⟩`}`
Scale *text* in horizontally by a factor *dima*/*dimb*, and vertically by a factor of *dimc*/*dimd*.

`\Gscale@div` `{`⟨*cmd*⟩`}{`⟨*dima*⟩`}{`⟨*dimb*⟩`}`
Define the macro *cmd* to be the ratio of the lengths *dima*/*dimb*.

6 Interface to the Driver Files

6.1 Graphics Inclusion

Each driver file must declare that its driver can include graphics of certain *types*. It does this by declaring for each type a command of the form:
`\Ginclude@`*type*
The Graphics kernel function will call this driver-defined function with the filename as argument, and certain additional information will be provided as follows.:

`\Gin@llx`, `\Gin@lly`, `\Gin@urx`, `\Gin@ury`	Macros storing the 'bounding box'
`\Gin@nat@width` `\Gin@nat@height`	Registers storing the natural size.
`\Gin@req@width` `\Gin@req@height`	Registers storing the required size, after scaling.
`\Gin@scalex`, `\Gin@scaley`	macros with the scale factors. A value of `!` means: Scale by the same amount as the other direction.
`\ifGin@clip`	`\newif` token, true if the graphic should be 'clipped' to the bounding box.

Optionally the driver may define a command of the form:
`\Gread@`*type*
This is responsible for reading an external file to find the bounding box information. If such a command is not declared, but a read-file is specified the command `\Gread@eps`, which is defined in the Graphics Kernel will be used.

6.2 Literal Postscript

Drivers that are producing PostScript output may want to define the following macros. They each take one argument which should be passed to an appropriate special. They are not used directly by this package but allow other packages to use the standard configuration file and package options to customise to various drivers:
`\Gin@PS@raw`, Literal PostScript special.
`\Gin@PS@restored`, Literal PostScript special, the driver will surround this with a save-restore pair.
`\Gin@PS@literal@header`, Postscript to be inserted in the header section of the PostScript file.
`\Gin@PS@file@header`, external file to be inserted in the header section of the PostScript file.

6.3 Rotation

`\Grot@start`, `\Grot@end` These macros must be defined to insert the appropriate `\special` to rotate the text between them by `\Grot@angle` degrees. The kernel function will make sure that the correct TEX spacing is produced, these functions only need insert the `\special`.

6.4 Scaling

`\Gscale@start`, `\Gscale@end`, as for rotation, but here scale the text by `\Gscale@x` and `\Gscale@y`.

7 Implementation

1 ⟨*package⟩

7.1 Initialisation

`\Gin@codes` First we save the catcodes of some characters, and set them to fixed values whilst this file is being read.

```
2 \edef\Gin@codes{%
3  \catcode`\noexpand\^^A\the\catcode`\^^A\relax
4  \catcode`\noexpand\"\the\catcode`\"\relax
5  \catcode`\noexpand\*\the\catcode`\*\relax
6  \catcode`\noexpand\!\the\catcode`\!\relax
7  \catcode`\noexpand\:\the\catcode`\:\relax}
```

```
\catcode`\^^A=\catcode`\%
\@makeother\"%
\catcode`\*=11
\@makeother\!%
\@makeother\:%
```

We will need to have an implementation of the trigonometric functions for the rotation feature. May as well load it now.

```
\RequirePackage{trig}
```

`\Grot@start` `\Grot@end` Initialise the rotation primitives.

```
\providecommand\Grot@start{\@latex@error{Rotation not supported}\@ehc
            \global\let\Grot@start\relax}
\providecommand\Grot@end{}
```

`\Gscale@start` `\Gscale@end` Initialise the scaling primitives.

```
\providecommand\Gscale@start{\@latex@error{Scaling not supported}\@ehc
            \global\let\Gscale@start\relax}
\providecommand\Gscale@end{}
```

`\Gread@BBox` `%%BoundingBox` as a macro for testing with `\ifx`. This may be redefined by the hiresbb option.

```
\edef\Gread@BBox{\@percentchar\@percentchar BoundingBox}
```

7.2 Options

`\ds@draft` `\ds@final`

```
\DeclareOption{draft}{\Gin@drafttrue}
\DeclareOption{final}{\Gin@draftfalse}
```

`\ifGin@draft` True in draft mode.

```
\newif\ifGin@draft
```

`\ds@hiresbb` If given this option the package 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).

```
\DeclareOption{hiresbb}{%
  \edef\Gread@BBox{\@percentchar\@percentchar HiResBoundingBox}}
```

`\ds@demo` If given this option the package will disregard the actual graphics file and insert a black box unless width or height are already specified.

```
\DeclareOption{demo}{%
  \AtBeginDocument{%
    \def\Ginclude@graphics#1{%
      \rule{\@ifundefined{Gin@@ewidth}{150pt}{\Gin@@ewidth}}%
      {\@ifundefined{Gin@@eheight}{100pt}{\Gin@@eheight}}}}}
```

`\Gin@driver` Driver in use.

```
\providecommand\Gin@driver{}
```

`\ds@dvips` `\ds@xdvi` Tomas Rockiki's PostScript driver (unix, MSDOS, VMS...). The `X11` previewer `xdvi` supports basically the same set of `\specials`.

```
\DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
\DeclareOption{xdvi}{\ExecuteOptions{dvips}}
```

`\ds@dvipdf` Sergey Lesenko's dvipdf driver.

```
\DeclareOption{dvipdf}{\def\Gin@driver{dvipdf.def}}
```

`\ds@dvipdfm` Mark Wick's dvipdfm driver.

```
\DeclareOption{dvipdfm}{\def\Gin@driver{dvipdfm.def}}
```

\ds@dvipdfmx The driver for the dvipdfmx project.

```
\DeclareOption{dvipdfmx}{\def\Gin@driver{dvipdfmx.def}}
```

\ds@xetex Jonathan Kew's TEX variant.

```
\DeclareOption{xetex}{\def\Gin@driver{xetex.def}}
```

\ds@pdftex Han The Thanh's TEX variant.

```
\DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}
```

\ds@dvipsone The drivers for the Y&Y TEX system.
\ds@dviwindo

```
\DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
\DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}
```

\ds@emtex Two freely available sets of drivers for MSDOS, OS/2 and Windows.
\ds@dviwin

```
\DeclareOption{emtex}{\def\Gin@driver{emtex.def}}
\DeclareOption{dviwin}{\def\Gin@driver{dviwin.def}}
```

\ds@oztex OzTEX (Macintosh). Since release 3 of OzTEX, merge with dvips back end.

```
\DeclareOption{oztex}{\ExecuteOptions{dvips}}
```

\ds@textures Textures (Macintosh).

```
\DeclareOption{textures}{\def\Gin@driver{textures.def}}
```

\ds@pctexps PCTEX (MSDOS/Windows) .
\ds@pctexwin
\ds@pctexhp
\ds@pctex32

```
\DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
\DeclareOption{pctexwin}{\def\Gin@driver{pctexwin.def}}
\DeclareOption{pctexhp}{\def\Gin@driver{pctexhp.def}}
\DeclareOption{pctex32}{\def\Gin@driver{pctex32.def}}
```

\ds@truetex Kinch TrueTeX, and its version with extended special support as shipped by Sci-
\ds@tcidvi entific Word.

```
\DeclareOption{truetex}{\def\Gin@driver{truetex.def}}
\DeclareOption{tcidvi}{\def\Gin@driver{tcidvi.def}}
```

\ds@vtex VTEX driver.

```
\DeclareOption{vtex}{\def\Gin@driver{vtex.def}}
```

\ds@dvi2ps If *anyone* is using any of these driver options would they let me know. All these
\ds@dvialw are essentially untried and untested as far as I know.
\ds@dvilaser
\ds@dvitops
\ds@psprint
\ds@pubps
\ds@ln

```
%\DeclareOption{dvi2ps}{\def\Gin@driver{dvi2ps.def}}
%\DeclareOption{dvialw}{\def\Gin@driver{dvialw.def}}
%\DeclareOption{dvilaser}{\def\Gin@driver{dvilaser.def}}
%\DeclareOption{dvitops}{\def\Gin@driver{dvitops.def}}
%\DeclareOption{psprint}{\def\Gin@driver{psprint.def}}
%\DeclareOption{pubps}{\def\Gin@driver{pubps.def}}
%\DeclareOption{ln}{\def\Gin@driver{ln.def}}
```

\ds@debugshow You probably don't want to use this...

```
\DeclareOption{debugshow}{\catcode`\^^A=9 \let\GDebug\typeout}
```

A local configuration file may define more options. It should also make one driver option the default, by calling `\ExecuteOptions` with the appropriate option.

```
\InputIfFileExists{graphics.cfg}{}{}
```

\ds@hiderotate

```
\DeclareOption{hiderotate}{%
  \def\Grot@start{\begingroup\setbox\z@\hbox\bgroup}
  \def\Grot@end{\egroup\endgroup}}
```

\ds@hidescale

```
\DeclareOption{hidescale}{%
  \def\Gscale@start{\begingroup\setbox\z@\hbox\bgroup}
  \def\Gscale@end{\egroup\endgroup}}
```

After the options are processed, load the appropriate driver file. If a site wants a default driver (eg `textures`) it just needs to put `\ExecuteOptions{textures}` in a `graphics.cfg` file.

```
\ProcessOptions
```

Check that a driver has been specified (either as an option, or as a default option in the configuration file). Then load the 'def' file for that option, if it has not already been loaded by some other package (for instance the color package).

```
\if!\Gin@driver!
  \PackageError{graphics}
    {No driver specified}
    {You should make a default driver option in a file \MessageBreak
     graphics.cfg\MessageBreak
     eg: \protect\ExecuteOptions{textures}%
     }
\else
  \PackageInfo{graphics}{Driver file: \Gin@driver}
  \@ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
\fi
```

7.3 Graphics Inclusion

This Graphics package uses a lot of dimension registers. TEX only has a limited number of registers, so rather than allocate new ones, re-use some existing LATEX registers. This is safe as long as all uses of the registers are *local*, and that you can be sure that you *never* need to have access to both uses within the same scope.

\Gin@llx
\Gin@lly
\Gin@urx
\Gin@ury

In fact these four lengths are now stored as macros not as dimen registers, mainly so that integer `bp` lengths may be passed exactly.

```
\def\Gin@llx{0}
\let\Gin@lly\Gin@llx
\let\Gin@urx\Gin@llx
\let\Gin@ury\Gin@llx
```

\Gin@nat@width
\Gin@nat@height

The 'natural' size of the graphic, before any scaling.

```
\let\Gin@nat@width\leftmarginv
\let\Gin@nat@height\leftmarginvi
```

\ifGin@clip

This switch is ⟨*true*⟩ if any graphics outside the specified bounding box (really viewport) should not be printed.

```
\newif\ifGin@clip
```

\DeclareGraphicsExtensions

Declare a comma separated list of default extensions to be used if the file is specified with no extension.

```
\newcommand\DeclareGraphicsExtensions[1]{%
  \edef\Gin@extensions{\zap@space#1 \@empty}}
```

\Gin@extensions

Initialise the list of possible extensions.

```
\providecommand\Gin@extensions{}
```

\includegraphics

Top level command for the standard interface, just look for a `*`.

```
\def\includegraphics{%
  \@ifstar
    {\Gin@cliptrue\Gin@i}%
    {\Gin@clipfalse\Gin@i}}
```

`\Gin@i` If an optional argument is present, call `\Gin@ii` to process it, otherwise call `\Ginclude@graphics`.

```
\def\Gin@i{%
  \@ifnextchar[%]
    \Gin@ii
    {\Gin@bboxfalse\Ginclude@graphics}}
```

`\Gin@ii` Look for a second optional argument.

```
\def\Gin@ii[#1]{%
  \@ifnextchar[%]
    {\Gin@iii[#1]}
    {\Gin@iii[0,0][#1]}}
```

`\Gin@iii` Set the cordinates of the **l**lower **l**eft corner, and the cordinates of the **u**pper **r**ight corner. The cordinates may be any TEX dimension, defaulting to `bp`.

```
\def\Gin@iii[#1,#2][#3,#4]#5{%
  \begingroup
  \Gin@bboxtrue
  \Gin@defaultbp\Gin@llx{#1}%
  \Gin@defaultbp\Gin@lly{#2}%
  \Gin@defaultbp\Gin@urx{#3}%
  \Gin@defaultbp\Gin@ury{#4}%
  \Ginclude@graphics{#5}%
  \endgroup}
```

`\Gin@defaultbp` `\Gin@def@bp` This macro grabs a length, `#2`, which may or may not have a unit, and if a unit is supplied, converts to 'bp' and stores the value in `#1`. If a unit is not supplied 'bp' is assumed, and `#2` is directly stored in `#1`. Note that supplying 'bp' is not quite the same as supplying no units, as in the former case a convertion via 'pt' and back to 'bp' takes place which can introduce rounding error. The error is invisibly small but files conforming to Adobe DSC should have *integer* Bounding Box Coordinates, and conceivably some drivers might demand integer values. (Although most seem to accept real values (if they accept bounding box coordinates at all) in the `\special`. This is the reason why the mechanism uses `\def` and not TEX lengths, as in earlier releases of the package.

```
\def\Gin@defaultbp#1#2{%
  \afterassignment\Gin@def@bp\dimen@#2bp\relax{#1}{#2}}
\def\Gin@def@bp#1\relax#2#3{%
   \if!#1!%
     \def#2{#3}%
    \else
      \dimen@.99626\dimen@
      \edef#2{\strip@pt\dimen@}%
    \fi}
```

`\DeclareGraphicsRule` Declare what actions should be taken for a particular file extension.
`#1` extension, `#2` type, `#3` read-file, `#4` command,

```
\def\DeclareGraphicsRule#1#2#3#4{%
  \edef\@tempa{\string *}\def\@tempb{#3}%
  \expandafter\edef\csname Gin@rule@#1\endcsname##1%
       {{#2}%
        {\ifx\@tempa\@tempb\noexpand\Gin@ext\else#3\fi}%
        {\ifx\indent#4\indent##1\else#4\fi}}}
```

An example rule base.

```
                  ext    type  read  command
\DeclareGraphicsRule{.ps}   {eps} {.ps}    {}
\DeclareGraphicsRule{.eps}  {eps} {.eps}   {}
\DeclareGraphicsRule{.ps.gz}{eps} {.ps.bb} {`zcat #1}
\DeclareGraphicsRule{.pcx}  {bmp} {}       {}
```

`\graphicspath` User level command to set the input path for graphics files. A list of directories, each in a `{}` group.

```
\def\graphicspath#1{\def\Ginput@path{#1}}
```

`\Ginput@path` The default graphic path is `\input@path`.

```
\ifx\Ginput@path\@undefined
  \let\Ginput@path\input@path
\fi
```

`\Gin@getbase` Given a possible extension, `#1`, check whether the file exists. If it does set `\Gin@base` and `\Gin@ext` to the filename stripped of the extension, and the extension, respectively.

```
\def\Gin@getbase#1{%
  \edef\Gin@tempa{%
    \def\noexpand\@tempa####1#1\space{%
      \def\noexpand\Gin@base{####1}}}%
  \IfFileExists{\filename@area\filename@base#1}%
    {\Gin@tempa
     \expandafter\@tempa\@filef@und
     \edef\Gin@ext{#1}}{}%
```

`\Gin@ext` Initialise the macro to hold the extension.

```
\let\Gin@ext\relax
```

`\Gin@sepdefault` This must match the token used by `\filename@parse` to delimit the extension.

```
\def\Gin@sepdefault{.}
```

`\Ginclude@graphics` The main internal function implementing graphics file inclusion. `#1` is the file name.

```
\def\Ginclude@graphics#1{%
  \begingroup
  \let\input@path\Ginput@path
  \filename@parse{#1}%
  \ifx\filename@ext\relax
    \@for\Gin@temp:=\Gin@extensions\do{%
      \ifx\Gin@ext\relax
        \Gin@getbase\Gin@temp
      \fi}%
  \else
    \Gin@getbase{\Gin@sepdefault\filename@ext}%
```

If the user supplied an explicit extension, just give a warning if the file does not exist. (It may be created later.)

```
    \ifx\Gin@ext\relax
       \@warning{File `#1' not found}%
       \def\Gin@base{\filename@area\filename@base}%
       \edef\Gin@ext{\Gin@sepdefault\filename@ext}%
    \fi
  \fi
```

If no extension is supplied, it is an error if the file does not exist, as there is no way for the system to know which extension to supply.

```
    \ifx\Gin@ext\relax
         \@latex@error{File `#1' not found}%
         {I could not locate the file with any of these extensions:^^J%
          \Gin@extensions^^J\@ehc}%
    \else
       \@ifundefined{Gin@rule@\Gin@ext}%
```

Handle default rule.

```
        {\ifx\Gin@rule@*\@undefined
           \@latex@error{Unknown graphics extension: \Gin@ext}\@ehc
         \else
           \expandafter\Gin@setfile\Gin@rule@*{\Gin@base\Gin@ext}%
          \fi}%
        {\expandafter\expandafter\expandafter\Gin@setfile
            \csname Gin@rule@\Gin@ext\endcsname{\Gin@base\Gin@ext}}%
  \fi
\endgroup}
```

`\ifGread@` True if a file should be read to obtain the natural size.

```
\newif\ifGread@\Gread@true
```

`\Gin@setfile` Set a file to the size specified in arguments, or in a 'read file'.

```
\def\Gin@setfile#1#2#3{%
  \ifx\\#2\\\Gread@false\fi
  \ifGin@bbox\else
    \ifGread@
      \csname Gread@%
         \expandafter\ifx\csname Gread@#1\endcsname\relax
           eps%
         \else
           #1%
         \fi
      \endcsname{\Gin@base#2}%
    \else
```

By now the natural size should be known either from arguments or from the file. If not generate an error. (The graphicx interface relaxes this condition slightly.)

```
      \Gin@nosize{#3}%
    \fi
  \fi
```

The following call will modify the 'natural size' if the user has supplied a viewport or trim specification. (Not available in the standard interface.)

```
  \Gin@viewport@code
```

Save the natural size, and then call `\Gin@req@sizes` whic (in the key-val interface) will calculate the required size from the natural size, and any scaling info.

```
  \Gin@nat@height\Gin@ury bp%
  \advance\Gin@nat@height-\Gin@lly bp%
  \Gin@nat@width\Gin@urx bp%
  \advance\Gin@nat@width-\Gin@llx bp%
  \Gin@req@sizes
```

Call `\Ginclude@`*type* to include the figure unless this is not defined, or draft mode is being used.

```
  \expandafter\ifx\csname Ginclude@#1\endcsname\relax
    \Gin@drafttrue
    \expandafter\ifx\csname Gread@#1\endcsname\relax
      \@latex@error{Can not include graphics of type: #1}\@ehc
      \global\expandafter\let\csname Gread@#1\endcsname\@empty
    \fi
  \fi
  \leavevmode
  \ifGin@draft
      \hb@xt@\Gin@req@width{%
        \vrule\hss
        \vbox to \Gin@req@height{%
           \hrule \@width \Gin@req@width
           \vss
           \edef\@tempa{#3}%
```

```
            \rlap{ \ttfamily\expandafter\strip@prefix\meaning\@tempa}%
            \vss
            \hrule}%
          \hss\vrule}%
  \else
```

Support `\listfiles` and then set the final box to the required size.

```
    \@addtofilelist{#3}%
    \ProvidesFile{#3}[Graphic file (type #1)]%
    \setbox\z@\hbox{\csname Ginclude@#1\endcsname{#3}}%
    \dp\z@\z@
    \ht\z@\Gin@req@height
    \wd\z@\Gin@req@width
  \box\z@
  \fi}
```

`\Gin@exclamation` Catcode 12 `!`, in case of French, or other language styles.

```
\def\Gin@exclamation{!}
```

`\Gin@req@sizes` `\Gin@scalex` `\Gin@scaley` `\Gin@req@height` `\Gin@req@width` In the standard interface there is no scaling, so the required size is the same as the natural size. In other interfaces `\Gin@req@sizes` will be responsible for setting these parameters. Here we can set them globally.

```
\let\Gin@req@sizes\relax
\def\Gin@scalex{1}%
\let\Gin@scaley\Gin@exclamation
\let\Gin@req@height\Gin@nat@height
\let\Gin@req@width\Gin@nat@width
```

`\Gin@viewport@code` In the standard interface there is no possibility of specifying a viewport, so this is a no-op.

```
\let\Gin@viewport@code\relax
```

`\Gin@nosize` This command is called in the case that the graphics type specifies no 'read file' and the user supplied no size arguments. In the standard interface can only generate an error.

```
\def\Gin@nosize#1{%
  \@latex@error
      {Cannot determine size of graphic in #1 (no size specifed)}%
      \@ehc}
```

7.4 Reading the BoundingBox in EPS files

`\ifGin@bbox` This switch should be set ⟨*true*⟩ once a size has been found, either in an argument, or in an external file.

```
\newif\ifGin@bbox
```

`\Gread@eps` Read an EPS file (`#1`) and search for a line starting with `%%BoundingBox` and returns the result by setting four dimension registers `\Gin@llx`, `\Gin@lly`, `\Gin@urx` and `\Gin@ury`.

```
\def\Gread@eps#1{%
  \begingroup
```

Make it reasonably safe to have binary headers in the EPS file before the bounding box line.

```
  \@tempcnta\z@
  \loop\ifnum\@tempcnta<\@xxxii
     \catcode\@tempcnta14 %
     \advance\@tempcnta\@ne
  \repeat
  \catcode`\^^?14 %
  \let\do\@makeother
  \dospecials
```

Make sure tab and space are accepted as white space.

```
\catcode`\ 10 %
\catcode`\^^I10 %
\catcode\endlinechar5 %
\@makeother\:%
\@makeother\-%
```

The first thing we need to do is to open the information file, if possible.

```
\immediate\openin\@inputcheck#1 %
\ifeof\@inputcheck
  \@latex@error{File `#1' not found}\@ehc
\else
```

Now we'll scan lines until we find one that starts with %%BoundingBox: We need to reset the catcodes to read the file, and so this is done in a group.

```
    \Gread@true
    \let\@tempb\Gread@false
    \loop
      \read\@inputcheck to\@tempa
      \ifeof\@inputcheck
        \Gread@false
      \else
        \expandafter\Gread@find@bb\@tempa:.\\%
      \fi
    \ifGread@
    \repeat
  \immediate\closein\@inputcheck
\fi
\ifGin@bbox\else
  \@latex@error
    {Cannot determine size of graphic in #1 (no BoundingBox)}%
    \@ehc
  \gdef\@gtempa{0 0 72 72 }%
\fi
\endgroup
\expandafter\Gread@parse@bb\@gtempa\\}
```

`\Gread@find@bb` If a line in the EPS file starts with a %%BoundingBox:, we will examine it more closely. Note using the 'extra' argument #2#3 causes any space after the : to be gobbled.

```
\long\def\Gread@find@bb#1:#2#3\\{%
  \def\@tempa{#1}%
  \ifx\@tempa\Gread@BBox
    \Gread@test@atend#2#3()\\%
  \fi}
```

`\Gread@test@atend` Determine if the stuff following the %%BoundingBox is '(atend)', which will involve further reading of the file. This is accomplished by making \@tempb into a no-op, so that finding a %%BoundingBox does not stop the loop.

```
\def\Gread@test@atend#1(#2)#3\\{%
  \def\@tempa{#2}%
  \ifx\@tempa\Gread@atend
    \Gread@true
    \let\@tempb\relax
  \else
    \gdef\@gtempa{#1}%
    \@tempb
    \Gin@bboxtrue
  \fi}
```

`\Gread@parse@bb` We have %%BoundingBox and what follows is not '(atend)' so we will parse the rest of the line as a BB with four elements. PostScript files should never have

units specified in the BoundingBox comment, but we allow arbitrary TEX units in external files, or in other interfaces.

```
\def\Gread@parse@bb#1 #2 #3 #4 #5\\{%
  \Gin@defaultbp\Gin@llx{#1}%
  \Gin@defaultbp\Gin@lly{#2}%
  \Gin@defaultbp\Gin@urx{#3}%
  \Gin@defaultbp\Gin@ury{#4}}%
```

`\Gread@atend` `atend` as a macro for testing with `\ifx`.

```
\def\Gread@atend{atend}
```

7.5 Rotation

As above, we will re-use some existing local registers.

`\Grot@height` `\Grot@left` `\Grot@right` `\Grot@depth` Final Rotated box dimensions

```
\let\Grot@height\@ovxx
\let\Grot@left\@ovyy
\let\Grot@right\@ovdx
\let\Grot@depth\@ovdy
```

`\Grot@h` `\Grot@l` `\Grot@r` `\Grot@d` Original box dimensions

```
\let\Grot@l\@ovro
\let\Grot@r\@ovri
\let\Grot@h\@xdim
\let\Grot@d\@ydim
```

`\Grot@x` `\Grot@y` Coordinates of centre of rotation.

```
\let\Grot@x\@linelen
\let\Grot@y\@dashdim
```

`\rotatebox` The angle is specified by `#1`. The box to be rotated is `#2`. In the standard interface the centre of rotation is $(0, 0)$. Then finally call `\Grot@box` to rotate the box.

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

`\Grot@setangle` Set the internal macro used by `\Grot@box`. In the standard interface this is trivial, but other interfaces may have more interesting definitions. For example:

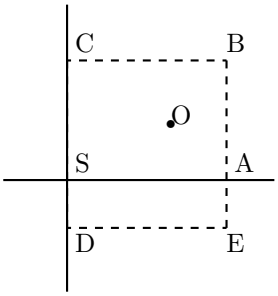
```
\def\Grot@setangle#1{%
  \dimen@#1\p@
  \dimen@-57.2968\dimen@
  \edef\Grot@angle{\strip@pt\dimen@}}
```

This would cause the argument of `\rotatebox` to be interpreted as an angle specified in *radians*, *clockwise*.

```
\def\Grot@setangle#1{\edef\Grot@angle{#1}}
```

7.6 Deriving a 'bounding box' for rotated object

We want to know the size of a 'bounding box' enclosing the rotated box. We define two formulae (as TEX macros) to work out the x and y coordinates of vertices of the rotated box in relation to its original coordinates (ie its width, height and depth). The box we visualize with vertices B, C, D and E is illustrated below. The vertex S is the reference point on the baseline. O is the centre of rotation, which in the standard interface is always S.



The formulae are, for a point P and angle α:

$$
\begin{aligned}
P'_x &= P_x - O_x \\
P'_y &= P_y - O_y \\
P''_x &= (P'_x \times \cos(\alpha)) - (P'_y \times \sin(\alpha)) \\
P''_y &= (P'_x \times \sin(\alpha)) + (P'_y \times \cos(\alpha)) \\
P'''_x &= P''_x + O_x + L_x \\
P'''_y &= P''_y + O_y
\end{aligned}
$$

The 'extra' horizontal translation L_x at the end is calculated so that the leftmost point of the resulting box has x-coordinate 0. This is desirable as TEX boxes must have the reference point at the left edge of the box.

`\Grot@Px` Work out new x coordinate of point after rotation. The parameters `#2` and `#3` are the original x and y coordinates of the point. The new x coordinate is stored in `#1`.

```
\def\Grot@Px#1#2#3{%
        #1\Grot@cos#2%
        \advance#1-\Grot@sin#3}
```

`\Grot@Py` Work out new y coordinate of point after rotation. The parameters `#2` and `#3` are the original x and y coordinates of the point. The new y coordinate is stored in `#1`.

```
\def\Grot@Py#1#2#3{%
        #1\Grot@sin#2%
        \advance#1\Grot@cos#3}
```

`\Grot@box` This is the tricky bit. We can rotate the box, but then need to work out how much space to leave for it on the page.

We simplify matters by working out first which quadrant we are in, and then picking just the right values.

```
\def\Grot@box{%
  \begingroup
```

We are going to need to know the sine and cosine of the angle; simplest to calculate these now.

```
  \CalculateSin\Grot@angle
  \CalculateCos\Grot@angle
  \edef\Grot@sin{\UseSin\Grot@angle}%
  \edef\Grot@cos{\UseCos\Grot@angle}%
^^A   \GDebug{Rotate: angle \Grot@angle, sine is \Grot@sin,
^^A             cosine is \Grot@cos}%
```

Save the four extents of the original box.

```
  \Grot@r\wd\z@  \advance\Grot@r-\Grot@x
  \Grot@l\z@     \advance\Grot@l-\Grot@x
  \Grot@h\ht\z@  \advance\Grot@h-\Grot@y
  \Grot@d-\dp\z@ \advance\Grot@d-\Grot@y
```

Now a straightforward test to see which quadrant we are operating in;

```
  \ifdim\Grot@sin\p@>\z@
    \ifdim\Grot@cos\p@>\z@
```

First quadrant: Height=By, Right=Ex, Left=Cx, Depth=Dy

```
      \Grot@Py\Grot@height \Grot@r\Grot@h%B
      \Grot@Px\Grot@right  \Grot@r\Grot@d%E
      \Grot@Px\Grot@left   \Grot@l\Grot@h%C
      \Grot@Py\Grot@depth  \Grot@l\Grot@d%D
    \else
```

Second quadrant: Height=Ey, Right=Dx, Left=Bx, Depth=Cy

```
      \Grot@Py\Grot@height \Grot@r\Grot@d%E
      \Grot@Px\Grot@right  \Grot@l\Grot@d%D
      \Grot@Px\Grot@left   \Grot@r\Grot@h%B
      \Grot@Py\Grot@depth  \Grot@l\Grot@h%C
    \fi
  \else
    \ifdim\Grot@cos\p@<\z@
```

Third quadrant: Height=Dy, Right=Cx, Left=Ex, Depth=By

```
      \Grot@Py\Grot@height \Grot@l\Grot@d%D
      \Grot@Px\Grot@right  \Grot@l\Grot@h%C
      \Grot@Px\Grot@left   \Grot@r\Grot@d%E
      \Grot@Py\Grot@depth  \Grot@r\Grot@h%B
    \else
```

Fourth quadrant: Height=Cy, Right=Bx, Left=Dx, Depth=Ey

```
      \Grot@Py\Grot@height \Grot@l\Grot@h%C
      \Grot@Px\Grot@right  \Grot@r\Grot@h%B
      \Grot@Px\Grot@left   \Grot@l\Grot@d%D
      \Grot@Py\Grot@depth  \Grot@r\Grot@d%E
    \fi
  \fi
```

Now we should translate back by (O_x, O_y), but TEX can not really deal with boxes that do not have the reference point at the left edge. (Everything with a $-$ve x-coordinate would over-print earlier text). So we modify the horizontal translation so that the reference point as understood by TEX *is* at the left edge. This means that the 'centre of rotation' is not fixed by `\rotatebox`, but typically moves horizontally. We also need to find the image of the original reference point, S, as that is where the rotation specials must be inserted.

```
  \advance\Grot@height\Grot@y
  \advance\Grot@depth\Grot@y
  \Grot@Px\dimen@  \Grot@x\Grot@y
  \Grot@Py\dimen@ii \Grot@x\Grot@y
  \dimen@-\dimen@     \advance\dimen@-\Grot@left
  \dimen@ii-\dimen@ii \advance\dimen@ii\Grot@y
^^A   \GDebug{Rotate: (l,r,h,d)^^J%
^^A Original \the\Grot@l,\the\Grot@r,\the\Grot@h,\the\Grot@d,^^J%
^^A New..... \the\Grot@left,\the\Grot@right,%
^^A          \the\Grot@height,\the\Grot@depth}%
  \setbox\z@\hbox{%
    \kern\dimen@
    \raise\dimen@ii\hbox{\Grot@start\box\z@\Grot@end}}%
  \ht\z@\Grot@height
  \dp\z@-\Grot@depth
  \advance\Grot@right-\Grot@left\wd\z@\Grot@right
  \leavevmode\box\z@
  \endgroup}
```

7.7 Stretching and Scaling

`\scalebox` The top level `\scalebox`. If the vertical scale factor is omitted it defaults to the horizontal scale factor, `#1`.

```
\def\scalebox#1{%
  \@ifnextchar[{\Gscale@box{#1}}{\Gscale@box{#1}[#1]}}
```

`\Gscale@box` Internal version of `\scalebox`.

```
\long\def\Gscale@box#1[#2]#3{%
  \leavevmode
  \def\Gscale@x{#1}\def\Gscale@y{#2}%
  \setbox\z@\hbox{{#3}}%
  \setbox\tw@\hbox{\Gscale@start\rlap{\copy\z@}\Gscale@end}%
  \ifdim#2\p@<\z@
    \ht\tw@-#2\dp\z@
    \dp\tw@-#2\ht\z@
  \else
    \ht\tw@#2\ht\z@
    \dp\tw@#2\dp\z@
  \fi
  \ifdim#1\p@<\z@
    \hb@xt@-#1\wd\z@{\kern-#1\wd\z@\box\tw@\hss}%
  \else
    \wd\tw@#1\wd\z@
    \box\tw@
  \fi}
```

`\reflectbox` Just an abbreviation for the appropriate scale to get reflection.

```
\def\reflectbox{\Gscale@box-1[1]}
```

`\resizebox` Look for a `*`, which specifies that a final vertical size refers to 'height + depth' not just 'height'.

```
\def\resizebox{%
  \leavevmode
  \@ifstar{\Gscale@@box\totalheight}{\Gscale@@box\height}}
```

`\Gscale@@box` Look for the `!` in the arguments.

```
\def\Gscale@@box#1#2#3{%
  \let\@tempa\Gin@exclamation
  \expandafter\def\expandafter\@tempb\expandafter{\string#2}%
  \expandafter\def\expandafter\@tempc\expandafter{\string#3}%
  \ifx\@tempb\@tempa
    \ifx\@tempc\@tempa
      \toks@{\mbox}%
    \else
      \toks@{\Gscale@box@dd{#3}#1}%
    \fi
  \else
    \ifx\@tempc\@tempa
      \toks@{\Gscale@box@dd{#2}\width}%
    \else
      \toks@{\Gscale@box@dddd{#2}\width{#3}#1}%
    \fi
  \fi
  \the\toks@}
```

`\Gscale@box@dd` Scale the text `#3` in both directions by a factor `#1`/`#2`.

```
\long\def\Gscale@box@dd#1#2#3{%
  \@begin@tempboxa\hbox{#3}%
    \setlength\@tempdima{#1}%
    \setlength\@tempdimb{#2}%
    \Gscale@div\@tempa\@tempdima\@tempdimb
    \Gscale@box\@tempa[\@tempa]{\box\@tempboxa}%
  \@end@tempboxa}
```

`\Gscale@box@dddd` Scale the text `#5` horizontally by a factor `#1`/`#2` and vertically by a factor `#3`/`#4`.

```
\long\def\Gscale@box@dddd#1#2#3#4#5{%
  \@begin@tempboxa\hbox{#5}%
```

```
    \setlength\@tempdima{#1}%
    \setlength\@tempdimb{#2}%
    \Gscale@div\@tempa\@tempdima\@tempdimb
    \setlength\@tempdima{#3}%
    \setlength\@tempdimb{#4}%
    \Gscale@div\@tempb\@tempdima\@tempdimb
    \ifGin@iso
      \ifdim\@tempa\p@>\@tempb\p@
        \let\@tempa\@tempb
      \else
        \let\@tempb\@tempa
      \fi
    \fi
    \Gscale@box\@tempa[\@tempb]{\box\@tempboxa}%
  \@end@tempboxa}
```

`\ifGin@iso` If this flag is true, then specifying two lengths to `\resizebox` scales the box by the same factor in either direction, such that neither length *exceeds* the stated amount. No user interface to this flag in the standard package, but it is used by the `keepaspectratio` key to `\includegraphics` in the graphicx package.

```
\newif\ifGin@iso
```

`\Gscale@div` The macro `#1` is set to the ratio of the lengths `#2` and `#3`.

```
\def\Gscale@div#1#2#3{%
  \setlength\dimen@{#3}%
  \ifdim\dimen@=\z@
    \PackageError{graphics}{Division by 0}\@eha
    \dimen@#2%
  \fi
  \edef\@tempd{\the\dimen@}%
  \setlength\dimen@{#2}%
  \count@65536\relax
  \ifdim\dimen@<\z@
    \dimen@-\dimen@
    \count@-\count@
  \fi
  \ifdim\dimen@>\z@
    \loop
      \ifdim\dimen@<8192\p@
        \dimen@\tw@\dimen@
        \divide\count@\tw@
    \repeat
    \dimen@ii\@tempd\relax
    \divide\dimen@ii\count@
    \divide\dimen@\dimen@ii
  \fi
  \edef#1{\strip@pt\dimen@}}
```

Restore Catcodes

```
\Gin@codes
\let\Gin@codes\relax
```

⟨/package⟩