

# 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`  $*[\langle llx, lly \rangle][\langle urx, ury \rangle]\{\langle file \rangle\}$   
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. If  $[\langle urx, ury \rangle]$  is present, then it should specify the coordinates of the top right corner of the image, as a pair of  $\text{\TeX}$  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  $[\langle llx, lly \rangle]$  may be used to specify the coordinates of this point.

`\graphicspath`  $\{\langle dir-list \rangle\}$

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  $\text{\LaTeX 2}_{\epsilon}$  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  $\text{\TeX}$  files are found.

`\DeclareGraphicsExtensions`  $\{\langle ext-list \rangle\}$

This specifies the behaviour of the system when the argument to `\includegraphics` does not have an extension specified. Here  $\{\langle ext-list \rangle\}$  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`  $\{\langle ext \rangle\}\{\langle type \rangle\}\{\langle read-file \rangle\}\{\langle command \rangle\}$

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  $\text{\TeX}$ , 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 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.

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  $\text{\LaTeX 2}_{\epsilon}$  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`  $\{\langle file \rangle\}$

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  $\langle true \rangle$  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`  $\{\langle file \rangle\}$

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`  $\{\langle xscale \rangle\}[\langle yscale \rangle]\{\langle text \rangle\}$

(The second argument is not optional.) Scale *text* by the appropriate scale factors.

`\Gscale@box@dd`  $\{\langle dima \rangle\}\{\langle dimb \rangle\}\{\langle text \rangle\}$

Scale *text* in both directions by a factor  $dima/dimb$ .

`\Gscale@box@ddd`  $\{\langle dima \rangle\}\{\langle dimb \rangle\}\{\langle dimc \rangle\}\{\langle dimd \rangle\}\{\langle text \rangle\}$

Scale *text* in horizontally by a factor  $dima/dimb$ , and vertically by a factor of  $dimc/dimd$ .

`\Gscale@div`  $\{\langle cmd \rangle\}\{\langle dima \rangle\}\{\langle dimb \rangle\}$

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.:

<code>\Gin@llx, \Gin@lly,</code> <code>\Gin@urx, \Gin@ury</code> <code>\Gin@nat@width</code> <code>\Gin@nat@height</code> <code>\Gin@req@width</code> <code>\Gin@req@height</code> <code>\Gin@scalex, \Gin@scaley</code>	Macros storing the ‘bounding box’  Registers storing the natural size.  Registers storing the required size, after scaling.  macros with the scale factors. A value of ! means: Scale by the same amount as the other direction.
<code>\ifGin@clip</code>	<code>\newif</code> 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\"the\relax
5  \catcode\`noexpand\*\the\catcode\*\relax
6  \catcode\`noexpand\!\the\catcode\!\relax
7  \catcode\`noexpand\:\the\catcode\:\relax}
```

```

8 \catcode`\^^A=\catcode`\%
9 \@makeother\"%
10 \catcode`\*=11
11 \@makeother\!%
12 \@makeother\:%

```

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

```

13 \RequirePackage{trig}

\Grot@start Initialise the rotation primitives.
\Grot@end
14 \providecommand\Grot@start{\@latex@error{Rotation not supported}\@ehc
15 \global\let\Grot@start\relax}
16 \providecommand\Grot@end{}

\Gscale@start Initialise the scaling primitives.
\Gscale@end
17 \providecommand\Gscale@start{\@latex@error{Scaling not supported}\@ehc
18 \global\let\Gscale@start\relax}
19 \providecommand\Gscale@end{}

\Gread@BBBox %%BoundingBox as a macro for testing with \ifx. This may be redefined by the
hiresbb option.
20 \edef\Gread@BBBox{\@percentchar\@percentchar BoundingBox}

```

## 7.2 Options

```

\ds@draft
\ds@final
21 \DeclareOption{draft}{\Gin@drafttrue}
22 \DeclareOption{final}{\Gin@draftfalse}

\ifGin@draft True in draft mode.
23 \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).
24 \DeclareOption{hiresbb}{%
25 \edef\Gread@BBBox{\@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.
26 \DeclareOption{demo}{%
27 \AtBeginDocument{%
28 \def\Gininclude@graphics#1{%
29 \rule{\@ifundefined{Gin@ewidth}{150pt}{\Gin@ewidth}}%
30 {\@ifundefined{Gin@eheight}{100pt}{\Gin@eheight}}}}

\Gin@driver Driver in use.
31 \providecommand\Gin@driver{}

\ds@dvips Tomas Rockiki's PostScript driver (unix, MSDOS, VMS...). The X11 previewer
\ds@xdvi xdvi supports basically the same set of \specials.
32 \DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
33 \DeclareOption{xdvi}{\ExecuteOptions{dvips}}

\ds@dvipdf Sergey Lesenko's dvipdf driver.
34 \DeclareOption{dvipdf}{\def\Gin@driver{dvipdf.def}}

\ds@dvipdfm Mark Wick's dvipdfm driver.
35 \DeclareOption{dvipdfm}{\def\Gin@driver{dvipdfm.def}}

```

```

\ds@dvipdfmx The driver for the dvipdfmx project.
               36 \DeclareOption{dvipdfmx}{\def\Gin@driver{dvipdfmx.def}}

\ds@xetex Jonathan Kew's TEX variant.
               37 \DeclareOption{xetex}{\def\Gin@driver{xetex.def}}

\ds@pdftex Han The Thanh's TEX variant.
               38 \DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}

\ds@dvipsone The drivers for the Y&Y TEX system.
\ds@dviwindo 39 \DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
               40 \DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}

\ds@emtex Two freely available sets of drivers for MSDOS, OS/2 and Windows.
\ds@dviwin 41 \DeclareOption{emtex}{\def\Gin@driver{emtex.def}}
               42 \DeclareOption{dviwin}{\def\Gin@driver{dviwin.def}}

\ds@oztex OzTEX (Macintosh). Since release 3 of OzTEX, merge with dvips back end.
               43 \DeclareOption{oztex}{\ExecuteOptions{dvips}}

\ds@textures Textures (Macintosh).
               44 \DeclareOption{textures}{\def\Gin@driver{textures.def}}

\ds@pctexps PCTEX (MSDOS/Windows) .
\ds@pctexwin 45 \DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
\ds@pctexhp 46 \DeclareOption{pctexwin}{\def\Gin@driver{pctexwin.def}}
\ds@pctex32 47 \DeclareOption{pctexhp}{\def\Gin@driver{pctexhp.def}}
               48 \DeclareOption{pctex32}{\def\Gin@driver{pctex32.def}}

\ds@truetetex Kinch TrueTEX, and its version with extended special support as shipped by Sci-
\ds@tcidvi entific Word.
               49 \DeclareOption{truetetex}{\def\Gin@driver{truetetex.def}}
               50 \DeclareOption{tcidvi}{\def\Gin@driver{tcidvi.def}}

\ds@vtex VTEX driver.
               51 \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@dviawl are essentially untried and untested as far as I know.
\ds@dvilaser 52 %\DeclareOption{dvi2ps}{\def\Gin@driver{dvi2ps.def}}
\ds@dvitops 53 %\DeclareOption{dviawl}{\def\Gin@driver{dviawl.def}}
\ds@psprint 54 %\DeclareOption{dvilaser}{\def\Gin@driver{dvilaser.def}}
\ds@pubps 55 %\DeclareOption{dvitops}{\def\Gin@driver{dvitops.def}}
\ds@ln 56 %\DeclareOption{psprint}{\def\Gin@driver{psprint.def}}
               57 %\DeclareOption{pubps}{\def\Gin@driver{pubps.def}}
               58 %\DeclareOption{ln}{\def\Gin@driver{ln.def}}

\ds@debugshow You probably don't want to use this...
               59 \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 op-
               tion.
               60 \InputIfFileExists{graphics.cfg}{\def\Gin@driver{graphics.def}}{}

\ds@hiderotate
               61 \DeclareOption{hiderotate}{%
               62 \def\Grot@start{\begingroup\setbox\z@\hbox\bgroup}
               63 \def\Grot@end{\egroup\endgroup}}

```

`\ds@hidescale`

```
64 \DeclareOption{hidescale}{%
65   \def\Gscale@start{\begingroup\setbox\z@\hbox\bgroup}
66   \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.

```
67 \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).

```
68 \if!\Gin@driver!
69   \PackageError{graphics}
70     {No driver specified}
71     {You should make a default driver option in a file \MessageBreak
72       graphics.cfg\MessageBreak
73       eg: \protect\ExecuteOptions{textures}}%
74   }
75 \else
76   \PackageInfo{graphics}{Driver file: \Gin@driver}
77   \@ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
78 \fi
```

### 7.3 Graphics Inclusion

This Graphics package uses a lot of dimension registers.  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  only has a limited number of registers, so rather than allocate new ones, re-use some existing  $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  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 In fact these four lengths are now stored as macros not as dimen registers, mainly
\Gin@lly so that integer bp lengths may be passed exactly.
\Gin@urx 79 \def\Gin@llx{0}
\Gin@ury 80 \let\Gin@lly\Gin@llx
          81 \let\Gin@urx\Gin@llx
          82 \let\Gin@ury\Gin@llx
```

```
\Gin@nat@width The ‘natural’ size of the graphic, before any scaling.
\Gin@nat@height 83 \let\Gin@nat@width\leftmarginv
                 84 \let\Gin@nat@height\leftmarginvi
```

```
\ifGin@clip This switch is  $\langle true \rangle$  if any graphics outside the specified bounding box (really
viewport) should not be printed.
85 \newif\ifGin@clip
```

```
\DeclareGraphicsExtensions Declare a comma separated list of default extensions to be used if the file is spec-
ified with no extension.
86 \newcommand\DeclareGraphicsExtensions[1]{%
87   \edef\Gin@extensions{\zap@space#1 \@empty}}
```

```
\Gin@extensions Initialise the list of possible extensions.
88 \providecommand\Gin@extensions{}
```

```
\includegraphics Top level command for the standard interface, just look for a *.
89 \def\includegraphics{%
90   \@ifstar
91     {\Gin@cliptrue\Gin@i}%
92     {\Gin@clipfalse\Gin@i}}
```



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

```
93 \def\Gin@i{%
94   \ifnextchar[%
95     \Gin@ii
96     {\Gin@bbboxfalse\Gin@bbboxtrue}}
```

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

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

`\Gin@iii` Set the coordinates of the lower left corner, and the coordinates of the upper right corner. The coordinates may be any TeX dimension, defaulting to bp.

```
101 \def\Gin@iii[#1,#2][#3,#4]#5{%
102   \begingroup
103   \Gin@bbboxtrue
104   \Gin@defaultbp\Gin@llx{#1}%
105   \Gin@defaultbp\Gin@lly{#2}%
106   \Gin@defaultbp\Gin@urx{#3}%
107   \Gin@defaultbp\Gin@ury{#4}%
108   \Gin@bbboxtrue{#5}%
109   \endgroup}
```

`\Gin@defaultbp` 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 conversion 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.

```
110 \def\Gin@defaultbp#1#2{%
111   \afterassignment\Gin@def@bp\dimen@#2bp\relax{#1}{#2}}
112 \def\Gin@def@bp#1\relax#2#3{%
113   \if!#1!%
114     \def#2{#3}%
115   \else
116     \dimen@.99626\dimen@
117     \edef#2{\strip@pt\dimen@}%
118   \fi}
```

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

```
119 \def\DeclareGraphicsRule#1#2#3#4{%
120   \edef\@tempa{\string *}\def\@tempb{#3}%
121   \expandafter\edef\csname Gin@rule@#1\endcsname##1%
122     {#2}%
123     {\ifx\@tempa\@tempb\noexpand\Gin@ext\else#3\fi}%
124     {\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.

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

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

```
126 \ifx\Ginput@path\undefined
127   \let\Ginput@path\input@path
128 \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.

```
129 \def\Gin@getbase#1{%
130   \edef\Gin@tempa{%
131     \def\noexpand\@tempa###1#1\space{%
132       \def\noexpand\Gin@base{###1}}}%
133   \IfFileExists{\filename@area\filename@base#1}%
134     {\Gin@tempa
135      \expandafter\@tempa\@filef@und
136      \edef\Gin@ext{#1}}{}}%
```

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

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

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

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

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

```
139 \def\Gin@include@graphics#1{%
140   \begingroup
141   \let\input@path\Ginput@path
142   \filename@parse{#1}%
143   \ifx\filename@ext\relax
144     \@for\Gin@temp:=\Gin@extensions\do{%
145       \ifx\Gin@ext\relax
146         \Gin@getbase\Gin@temp
147       \fi}%
148   \else
149     \Gin@getbase{\Gin@sepdefault\filename@ext}%
150   \ifx\Gin@ext\relax
151     \@warning{File `#1' not found}%
152     \def\Gin@base{\filename@area\filename@base}%
153     \edef\Gin@ext{\Gin@sepdefault\filename@ext}%
154   \fi
155 \fi
```

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

```
156   \ifx\Gin@ext\relax
157     \@latex@error{File `#1' not found}%
158     {I could not locate the file with any of these extensions:^^J%
159     \Gin@extensions^^J\@ehc}%
160   \else
161     \@ifundefined{Gin@rule@\Gin@ext}%
162     {\ifx\Gin@rule@\@undefined
163       \@latex@error{Unknown graphics extension: \Gin@ext}\@ehc
```

Handle default rule.

```
162     {\ifx\Gin@rule@\@undefined
163       \@latex@error{Unknown graphics extension: \Gin@ext}\@ehc
```

```

164         \else
165         \expandafter\Gin@setfile\Gin@rule@*{\Gin@base\Gin@ext}%
166         \fi}%
167     {\expandafter\expandafter\expandafter\Gin@setfile
168         \csname Gin@rule@\Gin@ext\endcsname{\Gin@base\Gin@ext}}%
169     \fi
170 \endgroup}

\ifGread@ True if a file should be read to obtain the natural size.
171 \newif\ifGread@\Gread@true

\Gin@setfile Set a file to the size specified in arguments, or in a ‘read file’.
172 \def\Gin@setfile#1#2#3{%
173     \ifx\\#2\\\Gread@false\fi
174     \ifGin@bbox\else
175         \ifGread@
176             \csname Gread@%
177                 \expandafter\ifx\csname Gread@#1\endcsname\relax
178                     eps%
179                 \else
180                     #1%
181                 \fi
182             \endcsname{\Gin@base#2}%
183         \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.)

```

184         \Gin@nosize{#3}%
185         \fi
186     \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.)

```

187 \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.

```

188 \Gin@nat@height\Gin@ury bp%
189 \advance\Gin@nat@height-\Gin@lly bp%
190 \Gin@nat@width\Gin@urx bp%
191 \advance\Gin@nat@width-\Gin@llx bp%
192 \Gin@req@sizes

```

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

```

193 \expandafter\ifx\csname Ginclude@#1\endcsname\relax
194     \Gin@drafttrue
195     \expandafter\ifx\csname Gread@#1\endcsname\relax
196         \@latex@error{Can not include graphics of type: #1}\@ehc
197         \global\expandafter\let\csname Gread@#1\endcsname\@empty
198     \fi
199 \fi
200 \leavevmode
201 \ifGin@draft
202     \hb@xt@\Gin@req@width{%
203         \vrule\hss
204         \vbox to \Gin@req@height{%
205             \hrule \@width \Gin@req@width
206             \vss
207             \edef\@tempa{#3}%
208             \rlap{ \ttfamily\expandafter\strip@prefix\meaning\@tempa}%
209             \vss
210             \hrule}%
211         \hss\vrule}%

```

```

212 \else
Support \listfiles and then set the final box to the required size.
213 \@addtofilelist{#3}%
214 \ProvidesFile{#3}[Graphic file (type #1)]%
215 \setbox\z@\hbox{\csname Gininclude@#1\endcsname{#3}}%
216 \dp\z@\z@
217 \ht\z@\Gin@req@height
218 \wd\z@\Gin@req@width
219 \box\z@
220 \fi}

```

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

```

221 \def\Gin@exclamation{!}

```

`\Gin@req@sizes` 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.

```

222 \let\Gin@req@sizes\relax
223 \def\Gin@scalex{1}%
224 \let\Gin@scaley\Gin@exclamation
225 \let\Gin@req@height\Gin@nat@height
226 \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.

```

227 \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.

```

228 \def\Gin@nosize#1{%
229   \@latex@error
230     {Cannot determine size of graphic in #1 (no size specified)}%
231   \@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.

```

232 \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`.

```

233 \def\Gread@eps#1{%
234   \begingroup

```

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

```

235   \@tempcnta\z@
236   \loop\ifnum\@tempcnta<\@xxxii
237     \catcode\@tempcnta14 %
238     \advance\@tempcnta\@ne
239   \repeat
240   \catcode`\^^?14 %
241   \let\do\@makeother
242   \dospecials

```

Make sure tab and space are accepted as white space.

```

243 \catcode`\ 10 %
244 \catcode`\^~I10 %
245 \catcode\endlinechar5 %
246 \@makeother\:%
247 \@makeother\-%

```

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

```

248 \immediate\openin\@inputcheck#1 %
249 \ifeof\@inputcheck
250   \@latex@error{File `#1' not found}\@ehc
251 \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.

```

252   \Gread@true
253   \let\@tempb\Gread@false
254   \loop
255     \read\@inputcheck to\@tempa
256     \ifeof\@inputcheck
257       \Gread@false
258     \else
259       \expandafter\Gread@find@bb\@tempa:.\%
260     \fi
261   \ifGread@
262   \repeat
263   \immediate\closein\@inputcheck
264 \fi

265 \ifGin@bbox\else
266   \@latex@error
267     {Cannot determine size of graphic in #1 (no BoundingBox)}%
268   \@ehc
269   \gdef\@gtempa{0 0 72 72 }%
270 \fi
271 \endgroup
272 \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.

```

273 \long\def\Gread@find@bb#1:#2#3\%
274 \def\@tempa{#1}%
275 \ifx\@tempa\Gread@BBBox
276   \Gread@test@atend#2#3()\%
277 \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.

```

278 \def\Gread@test@atend#1(#2)#3\%
279 \def\@tempa{#2}%
280 \ifx\@tempa\Gread@atend
281   \Gread@true
282   \let\@tempb\relax
283 \else
284   \gdef\@gtempa{#1}%
285   \@tempb
286   \Gin@bboxtrue
287 \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 T<sub>E</sub>X units in external files, or in other interfaces.

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

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

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

## 7.5 Rotation

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

`\Grot@height` Final Rotated box dimensions

```
\Grot@left 294 \let\Grot@height\@ovxx
```

```
\Grot@right 295 \let\Grot@left\@ovyy
```

```
\Grot@depth 296 \let\Grot@right\@ovdx
```

```
297 \let\Grot@depth\@ovdy
```

`\Grot@h` Original box dimensions

```
\Grot@l 298 \let\Grot@l\@ovro
```

```
\Grot@r 299 \let\Grot@r\@ovri
```

```
\Grot@d 300 \let\Grot@h\@xdim
```

```
301 \let\Grot@d\@ydim
```

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

```
\Grot@y 302 \let\Grot@x\@linelen
```

```
303 \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.

```
304 \long\def\rotatebox#1#2{%
```

```
305 \leavevmode
```

```
306 \Grot@setangle{#1}%
```

```
307 \setbox\z@\hbox{{#2}}%
```

```
308 \Grot@x\z@
```

```
309 \Grot@y\z@
```

```
310 \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@
```

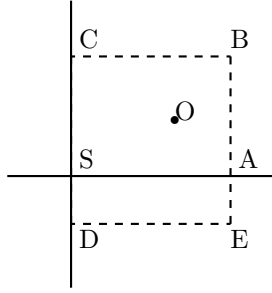
```
\edef\Grot@angle{\strip@pt\dimen@}}
```

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

```
311 \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 T<sub>E</sub>X 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  $\alpha$ :

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

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  $\text{\TeX}$  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.

```

312 \def\Grot@Px#1#2#3{%
313     #1\Grot@cos#2%
314     \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.

```

315 \def\Grot@Py#1#2#3{%
316     #1\Grot@sin#2%
317     \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.

```

318 \def\Grot@box{%
319     \begingroup

```

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

```

320 \CalculateSin\Grot@angle
321 \CalculateCos\Grot@angle
322 \edef\Grot@sin{\UseSin\Grot@angle}%
323 \edef\Grot@cos{\UseCos\Grot@angle}%
324 ^^A \GDebug{Rotate: angle \Grot@angle, sine is \Grot@sin,
325 ^^A cosine is \Grot@cos}%

```

Save the four extents of the original box.

```

326 \Grot@r\wd\z@ \advance\Grot@r-\Grot@x
327 \Grot@l\z@ \advance\Grot@l-\Grot@x
328 \Grot@h\ht\z@ \advance\Grot@h-\Grot@y
329 \Grot@d-\dp\z@ \advance\Grot@d-\Grot@y

```

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

```

330 \ifdim\Grot@sin\p@>\z@
331     \ifdim\Grot@cos\p@>\z@

```

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

```

332      \Grot@Py\Grot@height \Grot@r\Grot@h%B
333      \Grot@Px\Grot@right  \Grot@r\Grot@d%E
334      \Grot@Px\Grot@left   \Grot@l\Grot@h%C
335      \Grot@Py\Grot@depth  \Grot@l\Grot@d%D
336      \else

```

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

```

337      \Grot@Py\Grot@height \Grot@r\Grot@d%E
338      \Grot@Px\Grot@right  \Grot@l\Grot@d%D
339      \Grot@Px\Grot@left   \Grot@r\Grot@h%B
340      \Grot@Py\Grot@depth  \Grot@l\Grot@h%C
341      \fi
342      \else
343      \ifdim\Grot@cos\p@<\z@

```

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

```

344      \Grot@Py\Grot@height \Grot@l\Grot@d%D
345      \Grot@Px\Grot@right  \Grot@l\Grot@h%C
346      \Grot@Px\Grot@left   \Grot@r\Grot@d%E
347      \Grot@Py\Grot@depth  \Grot@r\Grot@h%B
348      \else

```

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

```

349      \Grot@Py\Grot@height \Grot@l\Grot@h%C
350      \Grot@Px\Grot@right  \Grot@r\Grot@h%B
351      \Grot@Px\Grot@left   \Grot@l\Grot@d%D
352      \Grot@Py\Grot@depth  \Grot@r\Grot@d%E
353      \fi
354      \fi

```

Now we should translate back by  $(O_x, O_y)$ , but  $\text{\TeX}$  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  $\text{\TeX}$  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.

```

355      \advance\Grot@height\Grot@y
356      \advance\Grot@depth\Grot@y
357      \Grot@Px\dimen@ \Grot@x\Grot@y
358      \Grot@Py\dimen@ii \Grot@x\Grot@y
359      \dimen@-\dimen@ \advance\dimen@-\Grot@left
360      \dimen@ii-\dimen@ii \advance\dimen@ii\Grot@y
361      ^^A \GDebug{Rotate: (l,r,h,d)}^^J%
362      ^^A Original \the\Grot@l,\the\Grot@r,\the\Grot@h,\the\Grot@d,^^J%
363      ^^A New.... \the\Grot@left,\the\Grot@right,%
364      ^^A \the\Grot@height,\the\Grot@depth}%
365      \setbox\z@\hbox{%
366      \kern\dimen@
367      \raise\dimen@ii\hbox{\Grot@start\box\z@\Grot@end}}%
368      \ht\z@\Grot@height
369      \dp\z@-\Grot@depth
370      \advance\Grot@right-\Grot@left\wd\z@\Grot@right
371      \leavevmode\box\z@
372      \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.

```

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

```



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

```
375 \long\def\Gscale@box#1[#2]#3{%
376   \leavevmode
377   \def\Gscale@x{#1}\def\Gscale@y{#2}%
378   \setbox\z@\hbox{#3}%
379   \setbox\tw@\hbox{\Gscale@start\rlap{\copy\z@}\Gscale@end}%
380   \ifdim#2\p<\z@
381     \ht\tw@-#2\dp\z@
382     \dp\tw@-#2\ht\z@
383   \else
384     \ht\tw@#2\ht\z@
385     \dp\tw@#2\dp\z@
386   \fi
387   \ifdim#1\p<\z@
388     \hb@xt@-#1\wd\z@{\kern-#1\wd\z@\box\tw@\hss}%
389   \else
390     \wd\tw@#1\wd\z@
391     \box\tw@
392   \fi}
```

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

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

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

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

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

```
397 \def\Gscale@@box#1#2#3{%
398   \let\@tempa\Gin@exclamation
399   \expandafter\def\expandafter\@tempb\expandafter{\string#2}%
400   \expandafter\def\expandafter\@tempc\expandafter{\string#3}%
401   \ifx\@tempb\@tempa
402     \ifx\@tempc\@tempa
403       \toks@{\mbox}%
404     \else
405       \toks@{\Gscale@box@dd{#3}#1}%
406     \fi
407   \else
408     \ifx\@tempc\@tempa
409       \toks@{\Gscale@box@dd{#2}\width}%
410     \else
411       \toks@{\Gscale@box@dddd{#2}\width{#3}#1}%
412     \fi
413   \fi
414   \the\toks@}
```

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

```
415 \long\def\Gscale@box@dd#1#2#3{%
416   \@begin@tempboxa\hbox{#3}%
417   \setlength\@tempdima{#1}%
418   \setlength\@tempdimb{#2}%
419   \Gscale@div\@tempa\@tempdima\@tempdimb
420   \Gscale@box\@tempa[\@tempa]{\box\@tempboxa}%
421   \@end@tempboxa}
```

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

```
422 \long\def\Gscale@box@dddd#1#2#3#4#5{%
423   \@begin@tempboxa\hbox{#5}%
424   \setlength\@tempdima{#1}%
425   \setlength\@tempdimb{#2}%
426   \setlength\@tempdimc{#3}%
427   \setlength\@tempdimd{#4}%
428   \Gscale@div\@tempa\@tempdima\@tempdimb\@tempdimc\@tempdimd
429   \Gscale@box\@tempa[\@tempa]{\box\@tempboxa}%
430   \@end@tempboxa}
```

```

425 \setlength\@tempdimb{#2}%
426 \Gscale@div\@tempa\@tempdima\@tempdimb
427 \setlength\@tempdima{#3}%
428 \setlength\@tempdimb{#4}%
429 \Gscale@div\@tempb\@tempdima\@tempdimb
430 \ifGin@iso
431 \ifdim\@tempa\p@>\@tempb\p@
432 \let\@tempa\@tempb
433 \else
434 \let\@tempb\@tempa
435 \fi
436 \fi
437 \Gscale@box\@tempa[\@tempb]{\box\@tempboxa}%
438 \@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.

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

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

```

440 \def\Gscale@div#1#2#3{%
441 \setlength\dimen@{#3}%
442 \ifdim\dimen@=\z@
443 \PackageError{graphics}{Division by 0}\@eha
444 \dimen@#2%
445 \fi
446 \edef\@tempd{\the\dimen@}%
447 \setlength\dimen@{#2}%
448 \count@65536\relax
449 \ifdim\dimen@<\z@
450 \dimen@-\dimen@
451 \count@-\count@
452 \fi
453 \ifdim\dimen@>\z@
454 \loop
455 \ifdim\dimen@<8192\p@
456 \dimen@\tw@\dimen@
457 \divide\count@\tw@
458 \repeat
459 \dimen@ii\@tempd\relax
460 \divide\dimen@ii\count@
461 \divide\dimen@\dimen@ii
462 \fi
463 \edef#1{\strip@pt\dimen@}}

```

Restore Catcodes

```

464 \Gin@codes
465 \let\Gin@codes\relax
466 </package>

```