Using Imported Graphics in LaTeX and pdfLaTeX

Keith Reckdahl
`epslatex at yahoo dot com`

Version 3.0.1
January 12, 2006

This document describes first how to import graphics into LaTeX documents and then covers a wide variety issues about their use. Readers can locate specific information by checking the Table of Contents starting on page 5 or the Index starting on page 122.

Importing graphics begins with specifying the graphicx package

```
\usepackage{graphicx}
```

and then using the `\includegraphics` command to insert the file

```
\includegraphics{file}
```

The `\includegraphics` command is covered in more detail in Section 7 on page 22.

This document is divided into the following five parts

Part I: Background Information

This part provides historical information and describes basic LaTeX terminology. It also

- The Encapsulated PostScript (EPS) format, differences between EPS and PS files, and methods for converting non-EPS graphics to EPS.
- The graphic formats that can be directly imported with pdfTeX (JPEG, PNG, PDF, MetaPost) are described.
- Freeware/Shareware graphics software is described.

Part II: The LaTeX Graphics Bundle

This part describes the commands in the graphics bundle which import, scale, and rotate graphics. This part covers much of the information in the graphics bundle documentation [7].

Part III: Using Graphics Inclusion Commands

This part describes how the graphics bundle commands are used to import, rotate, and scale graphics. Three situations where graphics inclusion is modified are also covered:

- Compressed EPS files and non-EPS graphic formats (TIFF, GIF, JPEG, PICT, etc.) can also be inserted on-the-fly when `dvips` is used with an operating system which supports pipes (such as Unix). When using other operating systems, the non-EPS graphics must be converted to EPS beforehand.

  Since neither LaTeX nor `dvips` has any built-in decompression or graphics-conversion capabilities, that software must be provided by the user.
- Since many graphics applications support only ASCII text, the PSfrag system allows text in EPS files to be replaced with LaTeX symbols or mathematical expressions.
- When an EPS graphic is inserted multiple times (such as a logo behind the text or in the page header) the final PostScript includes multiple copies of the graphics. When the graphics are not bitmapped, a smaller final PostScript file can be obtained by defining a PostScript command for the graphics.

Part IV: The figure Environment

There are several advantages to placing graphics in figure environments. Figure environments automatically number graphics, allowing them to be referenced or included in a table of contents. Since the figures can float to avoid poor page breaks, it is much easier to produce a professional-looking document.

In addition to general information about the figure environment, this section describes the following figure-related topics:

- How to customize the figure environment, such as adjusting figure placement, figure spacing, caption spacing, and adding horizontal line between the figure and the text. Caption formatting can also be customized, allowing users to modify the style, width, and font of captions.
- How to create marginal figures and wide figures which extend into the margins.
- How to produce figures with landscape orientation in a portrait document.
- How to place captions beside the figure instead of below or above the figure.
- For two-sided documents, how to ensure that a figure appears on an odd or even page. Also, how to ensure that two figures appear on facing pages.
- How to create boxed figures.

Part V: Complex Figures

This part describes how to construct complex figures that contain multiple graphics.

- How to form side-by-side graphics, side-by-side figures, and side-by-side subfigures.
- How to place a table next to a figure in the same float.
- How to stack multiple rows of figures.
- How to construct continued figures which can span multiple pages.

Where to Get this Document

This document is available in PDF and PostScript form as

CTAN/info/epslatex/english/epslatex.ps
CTAN/info/epslatex/english/epslatex.pdf

where CTAN can be replaced by any of the following CTAN (Comprehensive TEX Archive Network) sites and mirrors

England	ftp://ftp.tex.ac.uk/tex-archive/
Germany	ftp://ftp.dante.de/tex-archive/
Denmark	ftp://tug.org/tex-archive
France	ftp://ftp.loria.fr/pub/ctan
Russia	ftp://ftp.chg.ru/pub/TeX/CTAN
Vermont, USA	ftp://ctan.tug.org/tex-archive/
Florida, USA	ftp://ftp.cise.ufl.edu/pub/mirrors/tex-archive/
Utah, USA	ftp://ctan.math.utah.edu/tex-archive/
Korea	ftp://ftp.ktug.or.kr/tex-archive/
Japan	ftp://ftp.riken.go.jp/pub/tex-archive/
Hong Kong	ftp://ftp.comp.hkbu.edu.hk/pub/TeX/CTAN/
Singapore	ftp://ftp.nus.edu.sg/pub/docs/TeX/
New Zealand	ftp://elena.aut.ac.nz/pub/CTAN
Australia	ftp://ctan.unsw.edu.au/tex-archive/
India	http://mirror.gnowledge.org/ctan/
South Africa	ftp://ftp.sun.ac.za/CTAN/
Brazil	ftp://ftp.das.ufsc.br/pub/ctan/

A complete list of CTAN mirrors can be obtained from the CTAN.sites file at any CTAN site.

Jean-Pierre Drucbert's French translation of Version 2.0 of this document is available in PDF and PostScript as

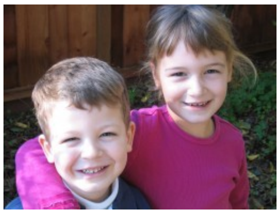
CTAN/info/epslatex/french/fepslatex.pdf
CTAN/info/epslatex/french/fepslatex.ps

Acknowledgments

I would like to thank David Carlisle for providing a great deal of assistance with this document. Donald Arseneau, Robin Fairbairns, Jim Hafner, Piet van Oostrum, Rolf Niepraschk, Axel Sommerfeldt, and other contributors to the comp.text.tex newsgroup provided much of the information for this document. Thanks to Jean-Pierre Drucbert for translating this document into French.

Thanks also goes to the many other people who provided me with valuable suggestions and bug reports for this document.

To my charming wife Becky,
and to Elise and Eric,
two wonderful kids who have disrupted and enriched my life
more than I ever could have imagined.



Contents

Part I
Background Information

1 Introduction

History When TEX was written, PostScript/EPS, JPEG, GIF, and other graphic formats did not exist. As a result, Knuth's DVI format does not have direct support for imported graphics. However, TEX allows DVI files to contain `\special` commands which pass commands to programs which use DVI files. This allowed TEX and LATEX to import any graphic format which is supported by the DVI program being used.

For many years, DVI files were usually converted to PostScript and the standard imported-graphic format was Encapsulated PostScript (EPS), which is a subset of the PostScript language. Inserting EPS graphics in LATEX originally required the low-level `\special` command. To make graphic-insertion easier and more portable, two higher-level packages epsf and psfig were written for LATEX2.09. In epsf, the graphics insertion was done by the `\epsfbox` command, while three other commands controlled graphic scaling. In psfig, the `\psfig` command not only inserted graphics, it also scaled and rotated them. While the psfig syntax was popular, its code was not as robust as epsf. As a result, the epsfig package was created as a hybrid of the two graphics packages, with its `\epsfig` command using the `\psfig` syntax and much of the more-robust `\epsfbox` code. Unfortunately, `\epsfig` still used some of the less-robust `\psfig` code.

LATEX Graphics Bundle With the release of LATEX 2ε in 1994, the LATEX3 team addressed the general problem of inserting graphics in LATEX. Their efforts produced the "LATEX graphics bundle[1] which contains totally re-written commands that are more efficient, more robust, and more portable than other graphics-insertion commands.

The graphics bundle contains the "standard" graphics package and the "extended" graphicx package. While both packages contain an `\includegraphics` command, the packages contain *different* versions of `\includegraphics`. The graphicx version uses "named arguments" (similar to the `\psfig` syntax) which, although convenient, violate the LATEX syntax guidelines which require that optional arguments be positional. As a compromise, two versions of `\includegraphics` were written, with the graphics package following the LATEX syntax guidelines and the graphicx package using the more-convenient named arguments. The graphicx `\includegraphics` supports scaling and rotating, but the graphics `\includegraphics` command must be nested inside `\rotatebox` or `\scalebox` commands to produce rotating or scaling.

This document uses the graphicx package because its syntax is more convenient than the graphics syntax. Since both packages have the same capabilities, the examples in this document can also be performed with the graphics package, although the resulting syntax may be more cumbersome and slightly less efficient. For a more-detailed description of the packages, see the graphics bundle documentation [7].

For backward-compatibility, the graphics bundle also includes the epsfig package which replaces the original LATEX 2ε epsfig package. The new epsfig package defines the `\epsfbox`, `\psfig`, and `\epsfig` commands as wrappers which simply call the `\includegraphics` command. Since these wrappers are less efficient than the straight `\includegraphics` command, the wrapped packaged should be used only

[1] Note that there is a plain TEX version of the LATEX graphics bundle. See the files in the directory `CTAN/macros/plain/graphics/`

for old documents, with `\includegraphics` used for all new documents.

Non-EPS Graphics In addition to improving EPS graphics-inclusion, the LaTeX graphics bundle also addressed the problem of including non-EPS graphic formats such as JPEG and GIF. Since DVI-to-PS converters generally did not support direct inclusion of most non-EPS formats, inserting these graphics into PostScript documents required the graphics to be converted into EPS ahead of time. While this ahead-of-time conversion is usually still the best approach, the graphics bundle provided another option: on-the-fly graphics conversion by the DVI-to-PS converter. Section 6.2 on page 17 describes graphics-conversion programs while Section 14 on page 42 describes how to use non-EPS graphics with DVI-to-PS converters.

pdfTeX When PostScript was the conventional final format for LaTeX documents, the process was a two-step procedure: (1) LaTeX was used to create a DVI file, and (2) a DVI-to-PS processor (such as `dvips`) was used to create a PostScript file. The advent and subsequent popularity of Adobe's PDF format initially added a third step to the conventional process: (3) a tool such as Ghostscript[2], Adobe Acrobat[3], or PStill[4] was used to convert the PostScript file to PDF.

However, not only was this three-step LaTeX-`dvips`-ghostscript process cumbersome, it made certain PDF features such as hyperlinks difficult to implement. To correct this, Hàn Thế Thành wrote a tool called TeX2PDF which modified the TeX engine to produce PDF files directly from TeX. TeX2PDF was eventually renamed pdfTeX and, with the help of many volunteers (and the blessing of Donald Knuth), was extended to implement the full typesetting capabilities of TeX. While pdfTeX nominally outputs PDF, it also has the capability of outputting the same DVI that would be produced by TeX.

Just as the `latex` command uses TeX to process LaTeX documents into DVI files, the command `pdflatex` uses pdfTeX to process LaTeX documents directly into PDF files.

pdfTeX and Graphics An important aspect of pdfTeX is its native inclusion of a variety of graphics formats: JPEG, PNG, PDF, MetaPost. Although older versions of pdfTeX supported native inclusion of TIFF files, the current version of pdfTeX does *not* support TIFF.

Also note that pdfTeX cannot not directly import EPS files[5], which requires users with EPS files to use a program like `epstopdf` which converts EPS files to PDF format, although this prevents the direct use of PSfrag (see Section 15 on page 45).

2 LaTeX Terminology

A *box* is any LaTeX object (characters, graphics, etc.) that is treated as a unit (see [1, page 103]). Each box has a *reference point* on its left side. The box's *baseline* is a horizontal line which passes through the reference point (see Figure 1). When LaTeX forms lines of text, characters are placed left-to-right with their reference points aligned on a horizontal line called the *current baseline*, aligning the characters' baselines with the current baseline. LaTeX follows the same process for typesetting graphics or other objects; the reference point of each object is placed on the current baseline.

[2]Free software, see Section 6.1 on page 17.

[3]Commercial software, see `www.adobe.com`

[4]Shareware, see `www.pstill.com`

[5]pdfTeX *can* directly import EPS files processed by PurifyEPS, see Section 5.5 on page 17.

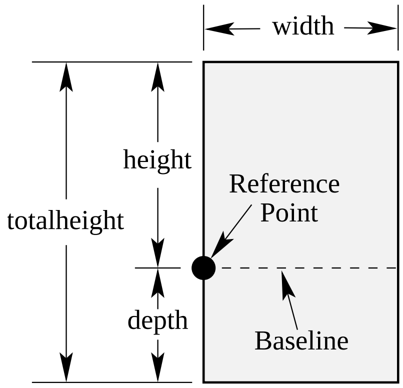


Figure 1: Sample LaTeX Box

The size of each box is described by three lengths: *height*, *depth*, *width*. The *height* is the distance from the reference point to the top of the box. The *depth* is the distance from the reference point to the bottom of the box. The *width* is the width of the box. The *totalheight* is defined as the distance from the bottom of the box to the top of the box, or totalheight=height+depth.

The reference point of a non-rotated EPS graphic is its lower-left corner (see left box in Figure 2), giving it zero depth and making its totalheight equal its height. The middle box in Figure 2 shows a rotated graphic where the height is not equal to the totalheight. The right box in Figure 2 shows a rotated graphic where the height is zero.

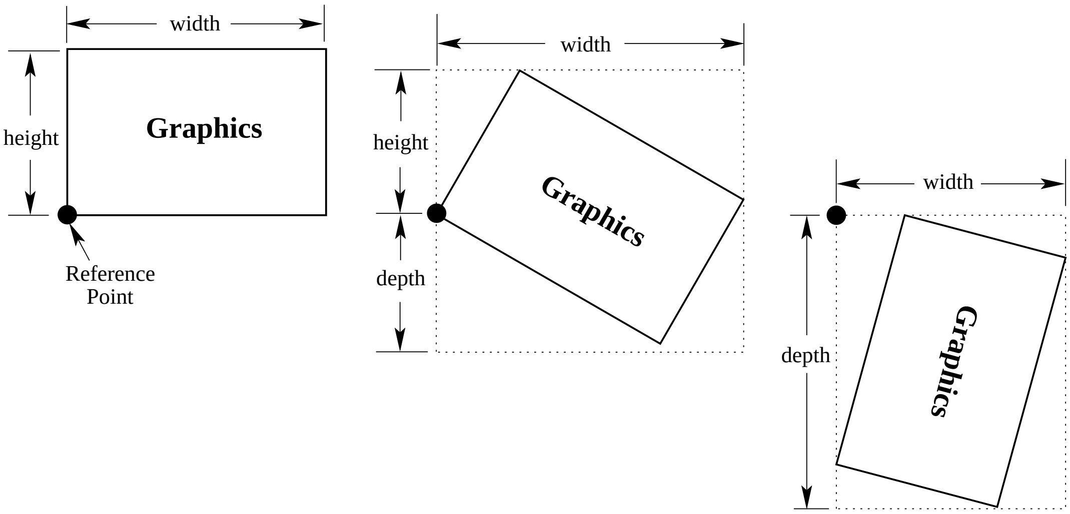


Figure 2: Rotated LaTeX Boxes

3 Encapsulated PostScript

The PostScript language describes both graphics and text. The PostScript language is used in conventional PostScript (PS) files to describe multiple-page documents and also in Encapsulated PostScript (EPS) files to describe graphics for insertion into documents. There are two main differences between PS and EPS files

- EPS files can contain only certain PostScript operators.

- EPS files must contain a BoundingBox line which specifies the size of the EPS graphic.

3.1 Forbidden PostScript Operators

Since EPS graphics must share the page with other objects, the commands in an EPS file cannot perform page operations such as selecting a page size (such as `letter` or `a4`) or erasing the entire page with `erasepage`. The following PostScript operators are not allowed in EPS files:

`a3`	`a4`	`a5`	`banddevice`	`clear`
`cleardictstack`	`copypage`	`erasepage`	`exitserver`	`framedevice`
`grestoreall`	`initclip`	`initgraphics`	`initmatrix`	`letter`
`legal`	`note`	`prenderbands`	`quit`	`renderbands`
`setdevice`	`setglobal`	`setpagedevice`	`setpageparams`	`setsccbatch`
`setshared`	`startjob`	`stop`		

Although the following PostScript operators can be used in EPS files, they may cause problems if not used properly.

`nulldevice`	`setcolortransfer`	`setgstate`	`sethalftone`
`setmatrix`	`setscreen`	`settransfer`	`undefinedfont`

Some of the above operators may cause the DVI-to-PS process to fail, while others may cause strange problems such as graphics which are misplaced, invisible, or flash on the screen. Since many of these operators do not affect the PostScript stack, such problems can often be eliminated by simply deleting the offending PostScript operator. Other cases may require complicated hacking of the PostScript code.

3.2 The EPS BoundingBox

By convention, the first line of a PostScript file specifies the type of PostScript and is then followed by a series of comments called the *header* or *preamble*. (Like LaTeX, PostScript's comment character is `%`). One of these comments specifies the BoundingBox. The BoundingBox line contains four integers

1. The x-coordinate of the lower-left corner of the BoundingBox.
2. The y-coordinate of the lower-left corner of the BoundingBox.
3. The x-coordinate of the upper-right corner of the BoundingBox.
4. The y-coordinate of the upper-right corner of the BoundingBox.

For example, the first 5 lines of an EPS file created by gnuplot are

```
%!PS-Adobe-2.0 EPSF-2.0
%%Creator: gnuplot
%%DocumentFonts: Times-Roman
%%BoundingBox: 50 50 410 302
%%EndComments
```

Thus the gnuplot EPS graphic has a lower-left corner with coordinates $(50, 50)$ and an upper-right corner with coordinates $(410, 302)$. The BoundingBox parameters have units of PostScript points which are $1/72$ of an inch, making the above graphic's natural width 5 inches and its natural height 3.5 inches. Note that a PostScript point is is slightly larger than a TeX point, which is $1/72.27$ of an inch. In TeX and LaTeX, PostScript points are called "big points" and abbreviated `bp` while TeX points are called "points" and abbreviated `pt`.

3.3 Converting PS files to EPS

Single-page PostScript files without any improper commands can be converted to EPS by using one of the following methods for adding a BoundingBox line. **Since these methods do not check for illegal PostScript operators, they do not produce usable EPS files unless the PS files are free of forbidden operators.**

1. The most convenient option is to use the `ps2epsi` utility distributed with Ghostscript (see Section 6.1 on page 17), which reads the PostScript file, calculates the BoundingBox parameters, and creates an EPS file (complete with a BoundingBox) which contains the PostScript graphics.

   The resulting file EPS file is in EPSI format, which means it contains an Interchange (low-resolution bitmapped) preview at the beginning of the file. Since this preview is ASCII-encoded, it does not cause the Section 4.1 `bufsize` errors. However, this EPSI preview increases the file size.

2. Another method of having ghostscript calculate the BoundingBox parameters is to use the `epstool` utility, available for Unix, DOS, Windows, and OS/2 from

   http://www.cs.wisc.edu/~ghost/gsview/epstool.htm

   For example, the command

   ```
   epstool --copy --bbox file1.eps file2.eps
   ```

   analyzes the contents of `file1.eps` to determine the correct BoundingBox and then copies the contents of `file1.eps` with the calculated BoundingBox into `file2.eps`.

   The `epstool` utility can also be used to add TIFF, WMF, EPSI bitmap previews to an EPS file, or extract bitmap previews from an EPS file.

3. Alternatively, the BoundingBox parameters can be calculated and manually inserted in the PostScript file's BoundingBox line or specified in the graphics-insertion command (e.g., the `\includegraphics` command's `bb` option). There are several ways to calculate the BoundingBox parameters

   (a) Use Ghostview/GSview to display the PostScript graphic. As the pointer is moved around the graphic, the pointer's coordinates (with respect to the lower-left corner of the page) are displayed. To determine the BoundingBox parameters, record the pointer coordinates at the lower-left corner of the graphic and the upper-right corner of the graphic.

   (b) Print out a copy of the PostScript graphics and measure the horizontal and vertical distances (in inches) from the lower-left corner of the paper to the lower-left corner of the graphics. Multiply these measurements by 72 to get the BoundingBox's lower-left coordinates. Likewise, measure the distances from the lower-left corner of the paper to the upper-right corner of the graphics to get the BoundingBox's upper-right coordinates.

   (c) The `bbfig` script uses a PostScript printer to calculate the BoundingBox. `bbfig` adds some PostScript commands to the beginning of the PostScript file and sends it to the printer. At the printer, the added PostScript commands calculate the BoundingBox of the original PostScript file, printing the BoundingBox coordinates superimposed on the PostScript graphic. The `bbfig` script is available from

   CTAN/support/bbfig/

3.4 Fixing Non-standard EPS files

Some applications (such as Mathematica and FrameMaker) produce non-standard EPS files which cannot be used in other programs such as LaTeX. Some of these applications have developed their own "improved" flavor of PostScript with additional features, while other applications use poor PostScript programming. Often these non-standard EPS can be easily fixed by scripts provided by either the software companies themselves or by PostScript-savvy users. Check the software manufacturer's web page or search USENET groups associated with the software.

4 How EPS Files are Used by LaTeX

When processing a `dvips`-style document, the EPS files are used by both LaTeX and the DVI-to-PS converter.

1. LaTeX scans the EPS file for the BoundingBox line, which tells LaTeX how much space to reserve for the graphic.
2. The DVI-to-PS converter then reads the EPS file and inserts the graphics in the PS file.

This has the following ramifications

- If the BoundingBox parameters are specified in the graphics-insertion command (e.g., the `bb` option of `\includegraphics` is used) then LaTeX never even reads the EPS file. In fact, the EPS file does not even need to exist when LaTeX is run.
- Since TeX cannot read non-ASCII files and cannot spawn other programs, LaTeX cannot read the BoundingBox information from compressed or non-EPS graphics files. In these cases, the BoundingBox parameters must be specified in the graphics-insertion command (e.g., in the `bb` option of the `\includegraphics` command) or stored in a non-compressed text file (see Section 14 on page 42).
- The EPS graphics are not included in the DVI file. Since the EPS files must be present when the DVI file is converted to PS, the EPS files must accompany DVI files whenever they are moved.
- The EPS graphics may not appear in some DVI viewers. To help the user with placement of the graphics, these DVI viewers generally display the BoundingBox in which the graphics will be inserted.

4.1 Line Buffer Overflow

TeX reads ASCII files one line at a time, putting each line in its line buffer, which is often about 3000 characters long. If any of the lines of the EPS file is longer than the line buffer, the following error is displayed

```
Unable to read an entire line--bufsize=3000.
Please ask a wizard to enlarge me.
```

Since EPS rarely have lines longer than 3000 characters, there are two possible causes of such an error

1. **The EPS file contains a long binary preview.**

   Some applications place a binary preview of the graphics at the beginning of the EPS file. This allows applications (such as DVI viewers) to display the graphics even though the application cannot interpret PostScript. Currently, relatively few TEX-related applications use such previews.

If the binary preview is smaller than the line buffer, the `\includegraphics` command skips over the preview[6]. However, the overfull `bufsize` error occurs if the binary preview is larger than the line buffer. There are a couple work-arounds for this problem

(a) If the preview won't be used, the problem can be avoided by either deleting it with a text editor or by preventing the original graphics application from creating the preview.

(b) Since LATEX reads the EPS file to only obtain the BoundingBox parameters, LATEX does not read the EPS file if the BoundingBox parameters are provided by the graphics-insertion command (e.g., the `bb` option to `\includegraphics`)

2. **The file's end-of-line characters are corrupted by an improper transfer.**

The following problem does not occur with most recent TEX distributions whose versions of TEX are smart enough to identify all end-of-line characters.

Different platforms use different end-of-line characters: Unix uses a line feed character (`^J`), Macintosh uses a carriage return (`^M`), while DOS/Windows uses a carriage return and line feed pair (`^M^J`). For example, if an EPS file is transferred in binary mode from a Macintosh to a Unix machine, the Unix TEX doesn't see any `^J` end-of-line characters and thus thinks the entire file is one big line, overfilling the line buffer.

If the EPS file has no binary sections (e.g., no binary preview and no embedded graphics) this problem can be avoided by transferring the EPS file in text mode. However, EPS files with binary sections must be transferred with binary mode, since the text mode transfer may corrupt the binary section. Since this binary transfer does not translate the end-of-line characters, the BoundingBox information must be provided by the graphics-insertion command (e.g., the `bb` option to `\includegraphics`).

5 PDF Graphics

As mentioned earlier, pdfTEX can directly import the PDF, PNG, JPEG, and MetaPost graphic formats. This section provides a short description of these formats. The commands for inserting these graphics into pdfLATEX are described in Section 7 on page 22.

5.1 JPEG

JPEG is a compression standard authored by Joint Photographic Experts Group (JPEG) Committee

`http://www.jpeg.org/`

[6]Note that `\psfig` and other obsolete graphics commands did not have the ability to skip over binary previews

The JPEG format is a compression standard for bitmap graphics which uses a lossy[7] compression scheme. In particular, its compression does not preserve lines and sharp edges, making it poorly suited for line drawings and or graphics with sharp features.

5.2 PNG

For many years the GIF format was the standard for compressed bitmaps for icons and other line drawings, since its lossless LZW compression does not distort sharp edges. Unisys's enforcement of its LZW patent coupled with some GIF technical limitations (such a limit of 256 colors) spurred the development of the Portable Network Graphics (PNG) format by a group eventually called the PNG Development Group

`http://www.libpng.org/pub/png/`

Like GIF, PNG uses lossless compression which is suitable for line drawings. While PNG can be used on any bitmap, JPEG's lossy compression is often better than PNG for photographs and other bitmaps without sharp edges (where "better" means producing smaller files without distortion noticeable by the naked eye).

5.3 PDF

Adobe's Portable Document Format (PDF) shares many similarities to its Adobe sibling PostScript. Like PostScript, PDF can contain text, vector drawings, and bitmap drawings. A PDF file can contain an entire document or just a single drawing (similar to EPS).

PDF is not only the primary output format of pdfTEX, but PDF also is the most common method for importing vector graphics into pdfTEX. Many graphics programs allow their graphics output to be directly saved in PDF format. Programs without direct PDF output can instead output EPS vector graphics which can be easily converted to PDF vector graphics by the `epstopdf` conversion program available from CTAN as a Windows executable or as a `perl` script for use on other platforms such as Unix/Linux or MacOS X

`CTAN/support/epstopdf/`

5.4 MetaPost

MetaPost is a graphics language written by John Hobby that is based on Donald Knuth's METAFONT, but with the added capability of outputting PostScript. Information about MetaPost is available from

`http://www.tug.org/metapost.html`
`http://cm.bell-labs.com/who/hobby/MetaPost.html`

and is documented in [25].

MetaPost can be used inn `dvips`-style LATEX documents and also can be used directly[8] by pdfLATEX documents.

[7] Lossy compression means that the compression process loses data. That is, decompresssing a lossy-compressed bitmap does *not* produce the original bitmap. Conversely, no data is lost during a *lossless* compression, so decompressing a lossless-compressed bitmap produces the original bitmap.

[8] pdfLATEX actually uses ConTeXt code by Hans Hagen to convert MetaPost graphics into PDF on-the-fly, however this is transparent to users.

The following procedure uses the pstoedit utility along with MetaPost (mpost) to convert an EPS file named graphic.eps into a MetaPost file name graphic.mps

```
pstoedit -f mpost graphic.eps graphic.mp
mpost graphic.mp
rename graphic.1 graphic.mps
```

5.5 PurifyEPS

Scott Pakin's purifyeps utility is able to convert many (but not all) EPS to a "purified" version that can be read by both LaTeX and pdfLaTeX.

You need all of the following in order to run purifyeps:

PurifyEPS Available from CTAN/support/purifyeps/ where CTAN/ should be replaced by any of the CTAN sites listed on page 3.

Perl Available from http://www.cpan.org

pstoedit Available from http://www.pstoedit.net/pstoedit

mpost from a LaTeX distribution that includes MetaPost.

6 Graphics Software

6.1 Ghostscript

Ghostscript is a PostScript/PDF interpreter which runs on most platforms and is distributed for free[9]. This allows PostScript, EPS, and PDF files to be displayed on the screen and printed to both Postscript and non-PostScript printers. AFPL Ghostscript is available from the Ghostscript home page

http://www.cs.wisc.edu/~ghost/

These sites contains pre-compiled Windows/DOS/OS/2 and Macintosh executables, along with ready-to-compile source code for Unix/VMS. Also available are graphical interfaces (such as GSview, Ghostview, GV, etc) for Ghostscript which makes the viewing of PostScript much easier.

6.2 Graphics-Conversion Programs

The following freeware and shareware programs convert between graphics format. In dvips-style documents, these programs can convert non-EPS graphics to EPS. In pdfLaTeX documents, these program can convert graphics to one of the supported formats (PDF, PNG, JPEG). Some of the programs allow command-line conversion which makes it possible to convert the graphics on-the-fly (see Section 14.2 on page 43).

[9]Although AFPL Ghostscript (formerly known as Aladdin Ghostscript) is distributed for free, it is not in the public domain. It is copyrighted and comes with certain limitations such as no commercial distribution. When versions of Aladdin Ghostscript become approximately one year old, Aladdin releases them as "GNU Ghostscript" whose use is governed by the less-restrictive GNU Public License.

ImageMagick

ImageMagick is a free open-source software suite to create, edit, and compose bitmap images. It can read, convert and write images in a large variety of formats. Images can be cropped, colors can be changed, various effects can be applied, images can be rotated and combined, and text, lines, polygons, ellipses and Bezier curves can be added to images and stretched and rotated.

For example, when ImageMagick's `convert` is on the operating system path, the following command

```
convert file.jpg file.eps
```

stores an EPS version of `file.jpg` in `file.eps`.

Multiple files can be converted with the wildcard

```
convert *.gif images.png
```

creates PNG versions of all the GIF files in the current directory and stores them as

```
images-0.png
images-1.png
...
```

Saving the resulting PNG files with the same base filename as the original GIF files is more involved since it requires writing a shell script or Windows batch file.

ImageMagick runs on all major operating systems and Binaries and information can be downloaded from

http://www.imagemagick.org/

GraphicsMagick

The ImageMagick interface periodically changes, causing incompatibility with code that uses ImageMagick. As a result, the GraphicsMagick project was started in November 2002 as a fork from ImageMagick 5.5.2, with the goal of providing a set of graphics-conversion utilities with a stable interface and an emphasis on fixing bugs over adding new features.

GraphicsMagick runs on Unix/Linux, Cygwin, MacOS X, and Windows. Binaries and source code can be downloaded from

http://www.graphicsmagick.org/

NetPBM

`NetPBM` is a free open-source version of the now-unsupported `PBMPLUS` package.

`NetPBM` is a toolkit for manipulation of graphic images, including conversion of images between a variety of different formats. There are over 220 separate tools in the package including converters for about 100 graphics formats. `NetPBM` uses the commandline and doesn't have a graphical interface.

Most Linux distributions and the Cygwin Project include `NetPBM` packages. Binary distributions of `NetPBM` for Windows, MacOS X, and other operating systems can be downloaded from http://netpbm.sourceforge.net/

Irfanview

Irfanview is an excellent, easy-to-install graphic viewer for Windows that is compact and fast. Irfanview supports viewing and converting between a wide variety of file formats, and provides basic image editing (such as cropping, resampling, color/brightness adjustments, etc). Irfanview supports both GUI and commandline operation, including batch mode.

For example, when Irfanview's executable `i_view32.exe` is on the Windows path, the following command

```
i_view32 *.gif /convert=*.png
```

creates PNG versions of all the GIF files in the current directory, storing them in files with `.png` extensions and the same base name as the original GIF files.

Irfanview can be downloaded from

http://www.irfanview.com/

Irfanview is freeware for personal, academic, and non-profit users. Commercial users are asked to donate a $12 registration fee.

Graphic Converter

Graphic Converter is $30 shareware for Macintosh which reads about 190 graphic formats and exports about 75 formats. For information, see

http://www.lemkesoft.de/

WMF2EPS

`WMF2EPS` is a $20 shareware WMF-to-EPS conversion program which runs on Windows. It is available from

CTAN/nonfree/support/wmf2eps/

where `CTAN/` should be replaced by any of the CTAN sites listed on page 3.

The software can also be downloaded from the `WMF2EPS` homepage

http://www.wmf2eps.de.vu/

The homepage also includes other information, including links to Adobe-compatible printer drivers (which is required for `WMF2EPS`).

KVEC

KVEC is shareware ($25 for non-commercial use, $50 for commercial use) which converts bitmap graphics (BMP, GIF, TIFF, etc) into PostScript and other vector formats. KVEC is available for Windows, OS/2, Linux, Unix, Macintosh, and BeOS. More information is available at

http://www.kvec.de

xv

xv is an interactive image manipulation program for the X Window System. While it has graphics-conversion capability, xv was designed for image manipulation program and thus is not tailored for graphics conversion (for example, it does not provide command-line capabilities so graphics must be one-by-one.

xv is $25 shareware for non-commercial use, with the $25 registration mandatory for commercial use. More information is available from

http://www.trilon.com/xv/xv.html

GIMP

GIMP (GNU Image Manipulation Program) is a freely available image manipulation program which duplicates much of the functionality of PhotoShop. GIMP is available for Unix/Linux, Windows, and MacOS X. More information is available at

http://www.gimp.org/

6.3 Level 2 EPS Wrappers

Level 2 PostScript supports several compression schemes, including DCT (used in JPEG files) and LZW (used in many TIFF files). Additionally, this binary data can be ASCII-encoded as ASCII85 or ASCIIHex (which produces ASCII files which are 125% and 200%, respectively, of the original binary size). The fact that Level-2 EPS supports these compression schemes allows a Level-2 EPS file to be constructed as a wrapper around a JPEG file or TIFF file. This produces better quality and smaller files than converting the graphics to conventional EPS. If one has a Level 2 PostScript printer, it is better to use the following wrapper programs instead of the conversion programs listed above. Since the resulting PostScript files can only be printed on Level 2 printers, the documents are less portable.

Note that, by default, `dvips` strips the comment lines (those lines which begin with `%%`) from any included EPS graphics. Since ASCII85-encoded level-2 graphics can have lines beginning with `%%`, users including ASCII85-encoded level-2 EPS files must use the `dvips -K0` (K followed by a zero) option to prevent `dvips` from stripping comment lines. Note ASCIIHex level-2 encoding does not have this problem.

jpeg2ps

A JPEG graphic can be converted to level 2 PostScript by the C program `jpeg2ps`, which can be compiled Unix, DOS, and other systems. `jpeg2ps` is available from

CTAN/nonfree/support/jpeg2ps/
http://www.pdflib.com/products/more/jpeg2ps.html
http://gnuwin32.sourceforge.net/packages/jpeg2ps.htm

where `CTAN/` should be replaced by any of the CTAN sites listed on page 3.

`jpeg2ps` supports three types of level-2 encoding: ASCII85 (default), 8-bit binary (using `jpeg2ps -b`), or 7-bit ASCIIHex (using `jpeg2ps -h`).

tiff2ps

A TIFF graphic can be converted to LZW-encoded Level-2 PostScript by using `tiff2ps`, which can be compiled on Unix, DOS, Mac, and VMS platforms. The source code for `tiff2ps` is available from the following three sites

http://www-mipl.jpl.nasa.gov/~ndr/tiff/html/tools.html
ftp://ftp.sgi.com/graphics/tiff/

ImageMagick's level-2 EPS Capability

As described in Section 6.2 on page 17, ImageMagick can convert between a large number of graphics formats. Since one of these formats is level-2 EPS, ImageMagick has the same functionality as the level-2 wrappers listed above. For example,

```
convert  file.jpeg  file.eps2
```

creates a level-2 EPS version of `file.jpeg` and stores it in `file.eps2`.

6.4 Editing PostScript

While the graphics in an EPS file can be modified by editing the file's PostScript commands, this is difficult for most people. Instead, it is easier to use the following programs to edit EPS graphics

pstoedit

`pstoedit` is a free program for Unix/Linux and Windows which converts PostScript and PDF graphics into vector formats (such as `Xfig`'s `.fig` format). More information is available at

http://www.pstoedit.com/

Mayura Draw

Mayura Draw (formerly known as PageDraw) is $39 shareware for Windows 3.1/95/NT which is available from

http://www.mayura.com/

When used with ghostscript, Mayura Draw can edit PostScript files.

xfig

`Xfig` is a free drawing program for Unix/Xwindow available from

http://www.xfig.org/

`Xfig` can import EPS drawings and add annotations, but currently cannot modify the original EPS graphics.

Part II
The LaTeX Graphics Bundle

This part provides an overview of The LaTeX Graphics Bundle. More detail can be found in the graphics bundle documentation [7] or the *LaTeX Graphics Companion* [4].

7 Graphics Inclusion

Graphics are imported using the `graphicx` package's `\includegraphics` command

Syntax: `\includegraphics[options]{filename}`

where the options are listed in Tables 1, 2, and 3. Since `\includegraphics` does not end the current paragraph, it can place graphics within text such as  or .

7.1 Graphics Driver

The user must specify a graphics driver which tells the graphics package how to process the imported graphic. The graphics bundle currently supports 18 different drivers, but this document only covers the two most common drivers: the `dvips` driver for `dvips`-style documents[10] and the `pdftex` driver for for pdfLaTeX documents. If the user wants to use either of these drivers, the driver usually does not need to be explicitly specified, as the `graphics.cfg` in most LaTeX distributions is smart enough to specify the correct driver[11].

Specifying A Driver

If the user needs to specify a driver, it can be specified in one of three ways

1. The default can be specified in the `graphics.cfg` file.
2. Any driver specified as a `\documentclass` option overrides the driver specified in `graphics.cfg`.
3. Any driver specified as an option in `\usepackage{graphics}` overrides the drivers specified in the previous two manners.

7.2 Graphics Inclusion for DVIPS-style Documents

The best-supported graphics format for `dvips`-style documents is EPS. When the document is processed with `latex`, the following command

```
\includegraphics{file.eps}
```

includes the graphics from the EPS file `file.eps` at its natural size. When the specified filename has no extension

```
\includegraphics{file}
```

then `\includegraphics` appends the extensions in the `\DeclareGraphicsExtensions` extension list (See Section 9.1 on page 29).

[10] Where `latex` processes the LaTeX file into a DVI file, which then is subsequently processed into PostScript form by `dvips`.

[11] The `graphics.cfg` file detects whether the document is being processed by `latex` or `pdflatex` and specifies a `dvips` option when for `latex` and a `pdftex` option for `pdflatex`.

7.3 Graphics Inclusion for pdfLaTeX Documents

pdfTeX supports the direct importing of PDF, PNG, JPEG, and MetaPost graphics. When the document is processed with `pdflatex`, the following commands

```
\includegraphics{file.pdf}

\includegraphics{file.png}

\includegraphics{file.jpg}

\includegraphics{file.mps}
```

include the graphics from the PDF file `file.pdf`, the PNG file `file.png`, and the JPEG file `file.jpg`, and the MetaPost file `file.mps` at their natural size. When the specified filename has no extension

```
\includegraphics{file}
```

then `\includegraphics` appends the extensions in the `\DeclareGraphicsExtensions` extension list (See Section 9.1 on page 29).

7.4 Documents to be Processed by both LaTeX and pdfLaTeX

It is often desired to allow a document to be processed by either LaTeX or pdfLaTeX, with LaTeX and `dvips` used when PostScript output is needed and pdfLaTeX to be used when PDF output is needed. Two things change when switching between LaTeX and pdfLaTeX:

- The appropriate graphicx driver changes.
- The graphic types that can be directly imported change.

The following steps adjust these things, allowing a document to be processed by either LaTeX or pdfLaTeX:

1. Create two copies[12] of each graphic to be imported:
   (a) An EPS version which is imported when `latex` processes the document.
   (b) A PNG, PDF, JPEG, or MetaPost version which is imported when `pdflatex` processes the document.
2. Do not specify `dvips` or `pdftex` as an option in the `\documentclass` or the `\usepackage{graphicx}` commands. Instead, the `graphic.cfg` command should automatically pass the appropriate option to the graphicx package.
3. When using the `\includegraphics` command to insert the graphics, do not specify any extension. For example:

   ```
   \includegraphics{graphic}
   ```

   The default extension list defined in `dvips.def` causes LaTeX to import the EPS version of the graphics while the default extension list defined in `pdftex.def` causes pdfLaTeX to import the PNG, PDF, JPEG, or MetaPost version of the graphics (see Section 9.1 on page 29).
4. Do not directly use PSfrag. If PSfrag substitution is needed, use the method described in Section 15.5 on page 48.

---

[12]Sometimes PurifyEPS (see Section 5.5 on page 17) can be used to create a single file that can be used by both LaTeX and pdfLaTeX.

7.4.1 Conditional Code with the ifpdf Package

The ifpdf package's `\ifpdf` command detects[13] whether the document is being processed by `latex` or `pdflatex`, allowing the document to have conditional code. For example, since it may be advisable to minimize the length of the extension list (as described in Section 9.1) the `\ifpdf` command can be used to customize the extension list

```
\usepackage{ifpdf}
...
\ifpdf
    \DeclareGraphicsExtensions{.pdf,.png,.jpg,.mps}
\else
    \DeclareGraphicsExtensions{.eps}
\fi
```

If the user wants the conditional code to use different `\documentclass` options, the follow code allows the `\ifpdf` command to be defined *before* the `\documentclass` command

```
\RequirePackage{ifpdf}
\ifpdf
   \documentclass[pdftex]{article}
\else
   \documentclass[dvips]{article}
\fi
```

This code passes the `[pdftex]` option if the document is being processed by `pdflatex` and passes the `[dvips]` option if the document is being processed by `latex`. As described in Section 7.1 on page 22, this code is generally not needed since most distributions automatically do this in their `graphics.cfg` file.

7.5 Specifying Width, Height, or Angle

Specifying Width

The command

```
\includegraphics[width=3in]{file}
```

includes the graphics from the specified file such that its width is 3 inches. Instead of specifying a fixed width (such as 3 inches) specifying the width in terms of scalable lengths[14] makes the graphic layout more robust. For example, the command

```
\includegraphics[width=\linewidth]{graphic}
```

scales the included graphic to be as wide as the current text. The command

```
\includegraphics[width=0.80\linewidth]{graphic}
```

makes the included graphic 80% as wide as the current text. When the calc package is used, the following command causes the graphics to be 2 inches more narrow than the current text

```
\includegraphics[width=\linewidth-2.0in]{graphic}
```

Specifying Height

Similarly, the command

[13] Historically, one method for doing this detection was to use the fact that `\pdfoutput` was defined only if pdfLaTeX was processing the document. However, many TeX distributions now have their `latex` command actually execute pdfLaTeX in DVI mode, causing `\pdfoutput` to be defined when both `latex` and `pdflatex` are executed. The ifpdf package solves this problem by providing a conditional command that robustly determines whether the document is being processed directly into a PDF file.

[14] The predefined scalable lengths are:

`\textwidth` is the width of the document's normal text.

`\linewidth` is the width of lines for the current environment.

`em` is the width of a capital M for the current font.

`ex` is the height of a lowercase x for the current font.

Table 1: includegraphics Options

`height`	The height of the graphics (in any of the accepted TeX units).
`totalheight`	The totalheight of the graphics (in any of the accepted TeX units).
`width`	The width of the graphics (in any of the accepted TeX units).
`scale`	Scale factor for the graphic. Specifying `scale=2` makes the graphic twice as large as its natural size.
`angle`	Specifies the angle of rotation, in degrees, with a counter-clockwise (anti-clockwise) rotation being positive.
`origin`	The `origin` command specifies what point to use for the rotation origin. By default, the object is rotated about its reference point. The possible origin points are the same as those for the `\rotatebox` command in Section 8.3 on page 28. For example, `origin=c` rotates the graphic about its center.
`bb`	Specifies BoundingBox parameters. For example `bb=10 20 100 200` specifies that the BoundingBox has its lower-left corner at (10,20) and its upper-right corner at (100,200). Since `\includegraphics` automatically reads the BoundingBox parameters from the EPS file, the `bb` option is usually not specified. It is useful if the BoundingBox parameters in the EPS file are missing or incorrect.

Table 2: includegraphics Cropping Options

`viewport`	Specifies what portion of the graphic to view. Like a BoundingBox, the area is specified by four numbers which are the coordinates of the lower-left corner and upper-right corner. The coordinates are relative to lower-left corner of the BoundingBox. For example, if the graphic's BoundingBox parameters are `50 50 410 302`, `viewport=50 50 122 122` displays the 1-inch square from the lower left of the graphic, and `viewport=338 230 410 302` displays the 1-inch square from the upper right of the graphic. The `clip` option (see Table 3) must be used to prevent the portion of the graphic outside the viewport from being displayed.
`trim`	An alternate method for specifying what portion of the graphic to view. The four numbers specify the amount to remove from the left, bottom, right, and top side, respectively. Positive numbers trim from a side, negative numbers add to a side. For example, `trim=1 2 3 4` trims the graphic by 1 `bp` on the left, 2 `bp` on the bottom, 3 `bp` on the right, 4 `bp` on the top. The `clip` option (see Table 3) must be used to prevent the trimmed portion from being displayed.

Table 3: includegraphics Boolean Options

`clip`	Specifying `clip=false` the entire graphic appears, even if portions appear outside the viewing area. (default) When `clip` or `clip=true` is specified, any graphics outside of the viewing area are clipped and do not appear.
`draft`	Specifying `draft` or `draft=true` prevents the graphic from being included in the document. The graphic's BoundingBox and filename are displayed in place of the graphic, making it faster to display and print the document. Specifying `draft=false` causes the EPS graphic to be inserted.
`keepaspectratio`	When `keepaspectratio` is not specified, specifying both the `width` and either `height` or `totalheight` causes the graphic to be scaled anamorphically to fit both the specified height and width. When `keepaspectratio` is specified, specifying both the `width` and either `height` or `totalheight` makes the graphic as large as possible such that its aspect ratio remains the same and the graphic exceeds neither the specified height nor width.

```
\includegraphics[height=2cm]{file}
```

includes the graphics from the specified file scaled such that its height is 2 cm. The `\includegraphics` command also includes a `totalheight` option for specifying a graphic's totalheight. (See Section 2 on page 10 for the definition of height and totalheight).

Specifying Angle The `\includegraphics` command's `angle` option specifies the angle of the included graphic

```
\begin{center}
  \includegraphics[angle=45]{graphic}
\end{center}
```

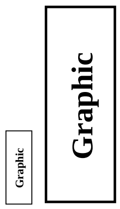
includes the graphic at its natural size and then rotates it by 45 degrees counter-clockwise (anti-clockwise).

7.5.1 Specifying Angle and Height or Width

Since the `\includegraphics` options are interpreted from left to right, the order in which the angle and size are specified makes a difference. For example

```
\begin{center}
  \includegraphics[angle=90,totalheight=1cm]{graphic}
  \includegraphics[totalheight=1cm,angle=90]{graphic}
\end{center}
```

produces



The first box is rotated 90 degrees and then scaled such that its height is one centimeter. The second box is scaled such that its height is one centimeter and then it is rotated 90 degrees.

Note that the two graphics in the above figure are separated by an interword space because the first `\includegraphics` line did not end with a `%`.

8 Rotating and Scaling Objects

In addition to the `\includegraphics` command, the graphicx package includes 4 other commands which rotate and scale *any* LaTeX object: text, EPS graphic, etc.

```
\scalebox{h-scale}[v-scale]{argument}
\resizebox{width}{height}{argument}
\resizebox*{width}{totalheight}{argument}
\rotatebox[options]{angle}{argument}
```

Since the graphicx `\includegraphics` command supports rotating and scaling options such as `angle` and `width`, the commands in this section rarely need to be used with EPS graphics. For example,

```
\includegraphics[scale=2]{file}
\includegraphics[width=4in]{file}
\includegraphics[angle=45]{file}
```

produce the same three graphics as

```
\scalebox{2}{\includegraphics{file}}
\resizebox{4in}{!}{\includegraphics{file}}
\rotatebox{45}{\includegraphics{file}}
```

However, the first syntax is preferred because it is faster and produces more efficient PostScript/PDF.

8.1 The scalebox Command

Syntax: `\scalebox{h-scale}[v-scale]{argument}`

The `\scalebox` command scales an object, making its width be `h-scale` times its original width and making the object's height be `v-scale` times its original height. If `v-scale` is omitted, it defaults to `h-scale`, keeping the aspect ratio constant. Negative values reflect the object.

8.2 The resizebox Commands

Syntax: `\resizebox{width}{height}{argument}`
`\resizebox*{width}{totalheight}{argument}`

The `\resizebox` command resizes an object to a specified size. Specifying `!` as either `height` or `width` makes that length be such that the aspect ratio remains constant. For example,

```
\resizebox{2in}{!}{argument}
```

scales the `argument` to be 2 inches wide while keeping its aspect ratio constant.

The standard LaTeX arguments `\height`, `\width`, `\totalheight`, `\depth` can be used to refer to the original size of `argument`. For example,

```
\resizebox{2in}{\height}{argument}
```

sizes argument to a width of 2 inches while keeping its same height.

The \resizebox* command is identical to \resizebox, except the second argument specifies the totalheight of the object. (See Section 2 on page 10 for the definition of height and totalheight, and see Section 11.1 on page 33 for a comparison of height and totalheight options.)

8.3 The rotatebox Command

Syntax: `\rotatebox[options]{angle}{argument}`

The \rotatebox command rotates an object by an angle given in degrees, with a counter-clockwise rotation being positive. By default, the object is rotated about its reference point. The \rotatebox options allow the point of rotation to be specified.

1. Specifying the [x=xdim,y=ydim], the object is rotated about the point whose coordinates relative to the reference point are (xdim,ydim).
2. The origin option specifies one of 12 special points shown in in Figure 3.

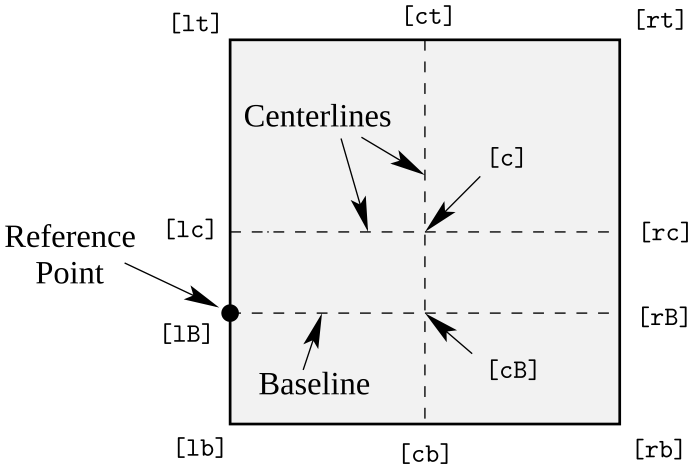


Figure 3: Available Origin Points

The horizontal position of the origin points is specified by one of three letters: l,c,r (which stand for left, center, right, respectively), while the vertical position is specified by one of four letters: t,c,B,b (which stand for top, center, Baseline, bottom, respectively). For example

[rb] specifies the bottom-right corner

[lt] specifies the top-left corner

[cB] specifies the center of the graphic's Baseline

Note that

- The order of the letters is not important, making [br] equivalent to [rb].
- c represents either the horizontal center or vertical center depending what letter is used with it.
- If only one letter is specified, the other is assumed to be c, making [c] equivalent to [cc], [l] equivalent to [lc], [t] equivalent to [tc], etc.

9 Advanced Graphics-Inclusion Commands

This section describes advanced graphics-inclusion commands which are needed in the following situations

1. When the specified filename has no extension. For example

   `\includegraphics{file}`

2. When compressed EPS graphics are used (also see Section 14.1 on page 43).
3. When non-EPS graphics are used (also see Section 14.2 on page 43).

In these situations, the `\DeclareGraphicsRule` and `\DeclareGraphicsExtensions` commands are needed to control how LaTeX handles files specified in `\includegraphics`.

- The `\DeclareGraphicsExtensions` command specifies the extensions to attempt (e.g., `.eps`, `.ps`, `.eps.gz`, etc.) when the specified filename does not have an extension.
- The `\DeclareGraphicsRule` specifies a command which operates on the file. The execution of this command requires an operating system which support pipes. For example, Unix supports pipes while DOS does not.

  Making this command a decompression command allows compressed EPS graphics to be used. Making this command a graphics-conversion command allows non-EPS graphics to be used.

9.1 The DeclareGraphicsExtensions Command

The `\DeclareGraphicsExtensions` command tells LaTeX which extensions to try if a file with no extension is specified in the `\includegraphics` command.

For convenience, a default set of extensions is pre-defined depending on which graphics driver is selected. For example if the graphicx package uses the `dvips` driver, the following graphic extensions (defined in `dvips.def`) are used by default

```
\DeclareGraphicsExtensions{.eps,.ps,.eps.gz,.ps.gz,.eps.Z}
```

If the graphicx package uses the `pdftex` driver, the following graphic extensions (defined in `pdftex.def`) are used by default

```
\DeclareGraphicsExtensions{.png,.pdf,.jpg,.mps}
```

With the `dvips` graphics extension list, `\includegraphics{file}` first looks for `file.eps`, then `file.ps`, then `file.eps.gz`, etc. until a file is found. This allows the graphics to be specified with

```
\includegraphics{file}
```

instead of

```
\includegraphics{file.eps}
```

The first syntax has the advantage that if you later decide to compress `file.eps`, you need not edit the LaTeX file. The extensionless syntax also allows the document to be processed by either LaTeX or pdfLaTeX as described in Section 7.4 on page 23.

However, the extensionless syntax can aggravate pool space problems as described in Section 9.1.2 below.

9.1.1 Filenames Without Extensions

Note that

```
\includegraphics{file}
```

does *not* attempt to open `file` unless the null extension `{}` is included in the extension list. For example,

```
\DeclareGraphicsExtensions{.eps,.eps.gz,{}}
```

causes `file` to be attempted if `file.eps` and `file.eps.gz` are not found.

Since the default extension list does *not* include the null extension, users wanting to use extensionless files must use the `\DeclareGraphicsExtensions` command to define an extension list that includes the null extension.

9.1.2 Pool Space Problems

Specifying no file extension and relying on LaTeX to choose the correct extension from the `\DeclareGraphicsExtensions` extension list can aggravate pool space problems (see Section 13.4 on page 41). If pool space is a concern, commands without an extension such as

```
\includegraphics{file}
```

should only be used when a `\DeclareGraphicsExtensions` command has specified a minimal number of extensions, such as

```
\DeclareGraphicsExtensions{.eps,.eps.gz}
```

9.2 The DeclareGraphicsRule Command

The `\DeclareGraphicsRule` command specifies how `\includegraphics` should treats file, depending on their extensions.

Syntax: `\DeclareGraphicsRule{ext}{type}{sizefile}{command}`

For example, the command

```
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}{`gunzip -c #1}
```

specifies that any file with a `.eps.gz` extension is treated as compressed EPS file, with the the BoundingBox information stored in the file with a `.eps.bb` extension, and the `gunzip -c` command uncompresses the file. (Since LaTeX cannot read BoundingBox information from a compressed file, the BoundingBox line must be stored in an uncompressed file.)

The `\DeclareGraphicsRule` allows `*` to signify any unknown extension. For example,

Table 4: DeclareGraphicsRule Arguments

`ext`	The file extension.
`type`	The graphics type for that extension.
`sizefile`	The extension of the file which contains the BoundingBox information for the graphics. If this option is blank, then the size information must be specified by the `\includegraphics` command's `bb` option.
`command`	The command to be applied to the file. (often left blank). The command must be preceded by a single backward quote (not to be confused with the more common single forward quote.) Currently, only `dvips` allows execution of such a command. See Section 14 on page 42 for examples of using this command for use with compressed and non-EPS graphics.

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

causes any unknown extension to be treated as an EPS file. For example, this causes file.EPS) to be treated as an EPS file.

Periods In Filenames The extension is defined as the portion of the filename after the first period, which makes it possible for files ending in `eps.gz` to be identified as compressed EPS files. To avoid confusion, the base portion of the filename should not contain a period. For example, specifying `file.name.eps.gz` makes `\includegraphics` look for a graphics rule associated with the extension `name.eps.gz`. Since such a graphics rule probably does not exist, the graphics rule for the unknown extension is used. (Filenames with multiple periods work if their type happens to be the default type. For example, when files with unknown extensions are treated as EPS, the filename `file.name.eps` is coincidently treated correctly.)

Pre-defined Commands For convenience, a default set of graphics rules is pre-defined depending on which graphics driver is selected. For example if the `dvips` driver is used, the following graphic rules (defined in `dvips.def`)[15] are used by default

```
\DeclareGraphicsRule{.eps}{eps}{.eps}{}
\DeclareGraphicsRule{.ps}{eps}{.ps}{}
\DeclareGraphicsRule{.pz}{eps}{.bb}{‘gunzip -c #1}
\DeclareGraphicsRule{.eps.Z}{eps}{.eps.bb}{‘gunzip -c #1}
\DeclareGraphicsRule{.ps.Z}{eps}{.ps.bb}{‘gunzip -c #1}
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}{‘gunzip -c #1}
\DeclareGraphicsRule{.ps.gz}{eps}{.ps.bb}{‘gunzip -c #1}
\DeclareGraphicsRule{.pcx}{bmp}{}{}
\DeclareGraphicsRule{.bmp}{bmp}{}{}
\DeclareGraphicsRule{.msp}{bmp}{}{}
\DeclareGraphicsRule{*}{eps}{*}{}
```

The first two commands define the `.eps` and `.ps` extensions as EPS files. The next five commands define extensions for compressed EPS files. The next three commands define extensions for PCX, BMP, MSP bitmaps. The last command causes filenames with unknown extensions to be treated as an EPS file.

If the pdfTEX driver is used, the following graphic rules[16] (defined in `pdftex.def`) are used by default

```
\DeclareGraphicsRule{.png}{png}{.png}{}
\DeclareGraphicsRule{.pdf}{pdf}{.pdf}{}
\DeclareGraphicsRule{.jpg}{jpg}{.jpg}{}
\DeclareGraphicsRule{.mps}{mps}{.mps}{}
```

which specify the behavior for pdfTEX's supported graphic formats.

[15]The code in `dvips.def` actually does not use `\DeclareGraphicsRule` but the effect is the same.

[16]The `pdftex.def` file actually checks the pdfTEX version and includes logic to define only the graphics rules which that version can handle

Part III
Using Graphics-Inclusion Commands

10 Horizontal Spacing and Centering

10.1 Horizontal Centering

The placement of the graphic is controlled by the current text justification. To center the graphic, put it inside a center environment

```
\begin{center}
   \includegraphics[width=2in]{graphic}
\end{center}
```

If the `\includegraphics` command is inside an environment (such as `minipage` or `figure`), the `\centering` declaration centers the remaining output of the environment. For example

```
\begin{figure}
   \centering
   \includegraphics[width=2in]{graphic}
\end{figure}
```

is similar to

```
\begin{figure}
\begin{center}
   \includegraphics[width=2in]{graphic}
\end{center}
\end{figure}
```

The `\centering` syntax is preferred because the `\begin{center}` syntax produces double vertical space above and below the figure due to the space produced by the `figure` environment and by the `center` environment. If extra vertical space is desired, the commands in Section 19.1 on page 64 should be used.

Obsolete Syntax Bugs in the `\psfig` and `\epsfbox` commands made it difficult to produce horizontally-centered graphics. The TeX commands `\centerline` and `\leavevmode` were used as work-arounds for bugs in `\psfig` and `\epsfbox`.

Since the `\includegraphics` command is written correctly, the `\centerline` and `\leavevmode` commands are no longer needed, allowing graphics to be centered with the `\centering` command or the `center` environment.

10.2 Horizontal Spacing

It is important to realize that LaTeX arranges graphics the same way it formats other objects such as letters. For example, an interword space is introduced between LaTeX input lines unless the line ends with a `%`. For example, just as

```
Hello
World
```

put an interword space between "Hello" and "World"

```
\includegraphics{file}
\includegraphics{file}
```

puts an interword space between the graphics. Ending the first line with a comment character

```
\includegraphics{file}%
\includegraphics{file}
```

puts no space between the graphics. When horizontal spacing is desired between graphics, the `\hspace` command inserts a specific amount of space[17] while `\hfill` inserts a rubber length which provides which expands to fill the available space. For example,

```
\includegraphics{file}\hfill\includegraphics{file}
```

pushes the graphics to the left and right margins, while

```
\hfill\includegraphics{file}%
\hfill\includegraphics{file}\hspace*{\fill}
```

puts equal spacing before, between, and after the graphics. Since `\hfill` commands which occur before a linebreak are ignored, the `\hspace*{\fill}` was needed to supply the trailing space.

Other Spacing Commands In addition to the `\hspace` and `\hfill` commands, the `\quad` command inserts horizontal space equal to the current font size (for example, when using 10 pt font, `\quad` inserts 10 pt of horizontal space). The command `\qquad` inserts twice as much horizontal space as `\quad`.

11 Rotation, Scaling, and Alignment

11.1 Difference Between Height and Totalheight

Care must be taken with the `height` option, as users often mean the overall height which is set by the `totalheight` option (see Figure 1 on page 11). When the object has zero depth, the `totalheight` is the same as the `height` and specifying `height` works fine. When the object has a non-zero depth, specifying `height` instead of `totalheight` causes either an incorrectly-sized graphic or a divide-by-zero error. For importing EPS files, the distinction between `height` and `totalheight` is most important when rotating and then scaling a graphic. For example,

```
\includegraphics[angle=-45,totalheight=1in]{file}
\includegraphics[angle=-45,height=1in]{file}
```

The first command scales the rotated graphic such that its total height is 1 inch. The second command scales the rotated graphics such that the portion above its reference point is 1 inch tall.

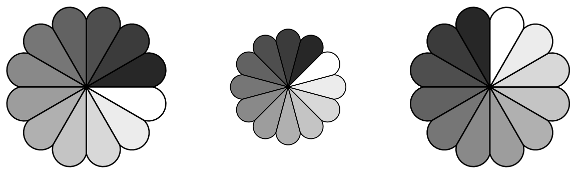
11.2 Scaling of Rotated Graphics

When the height or width of a graphic is specified, the specified size is not the size of the graphic but rather of its BoundingBox. This distinction is especially important in the scaling of rotated graphics. For example

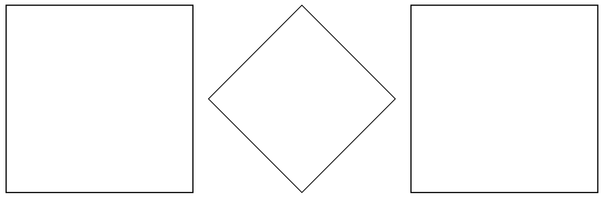
```
\begin{center}
  \includegraphics[totalheight=1in]{rosette}
  \includegraphics[angle=45,totalheight=1in]{rosette}
  \includegraphics[angle=90,totalheight=1in]{rosette}
\end{center}
```

produces

[17] Instead of making the `\hspace` amount a fixed length such as 1 inch, making the `\hspace` amount a function of `\linewidth` or `\em` increases a document's portability.



Although it may seem strange that the graphics have different sizes, it should make sense after viewing the BoundingBoxes



Each graphic is scaled such that its rotated BoundingBox is 1 inch tall. The scaling scales the size of the BoundingBox *not* the size of visible graphics.

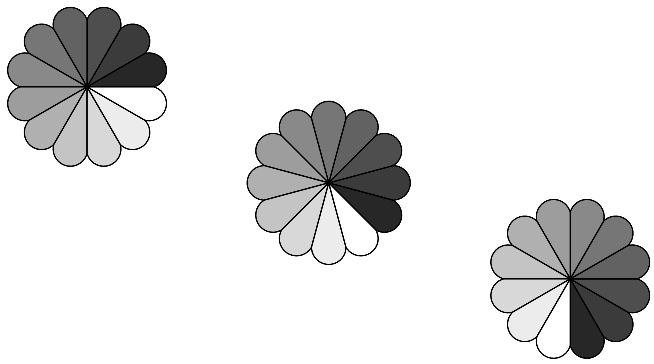
11.3 Alignment of Rotated Graphics

11.3.1 Example #1

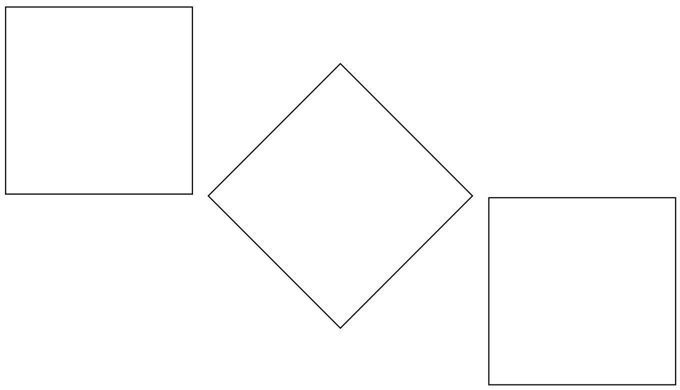
When graphics are rotated, the objects may not align properly. For example

```
\begin{center}
  \includegraphics[totalheight=1in]{rosette}
  \includegraphics[totalheight=1in,angle=-45]{rosette}
  \includegraphics[totalheight=1in,angle=-90]{rosette}
\end{center}
```

produces



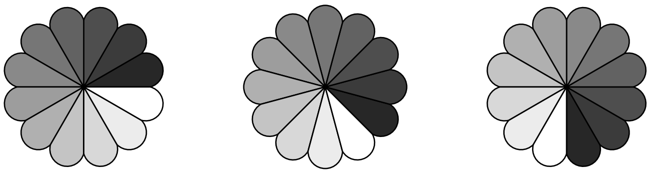
Again, this is better illustrated by the BoundingBoxes



In this case, the objects' reference points (original lower-left corners) are aligned on a horizontal line. If it is desired to instead have the centers aligned, the `origin` option of `\includegraphics` can be used

```
\begin{center}
  \includegraphics[totalheight=1in]{rosette}
  \includegraphics[totalheight=1in,origin=c,angle=-45]{rosette}
  \includegraphics[totalheight=1in,origin=c,angle=-90]{rosette}
\end{center}
```

This aligns the centers of the graphics

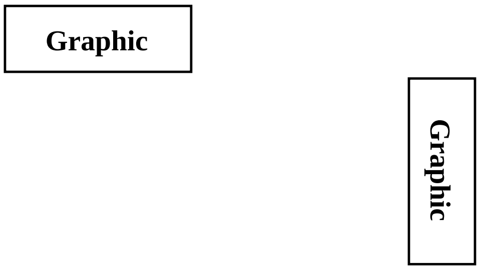


11.3.2 Example #2

Similarly, the commands

```
\begin{center}
    \includegraphics[width=1in]{graphic}
    \hspace{1in}
    \includegraphics[width=1in,angle=-90]{graphic}
\end{center}
```

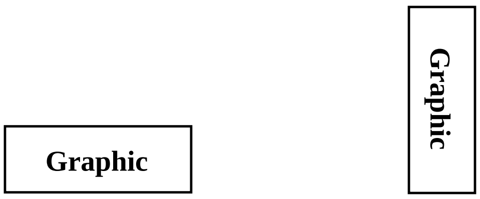
rotate the right-hand graphic around its lower-left corner, producing



To align the bottoms of the graphics, use the following commands

```
\begin{center}
   \includegraphics[width=1in]{graphic}
   \hspace{1in}
   \includegraphics[width=1in,origin=br,angle=-90]{graphic}
\end{center}
```

which rotate the right graphic about its lower-right corner, producing



11.4 Minipage Vertical Alignment

It is often useful to place graphics inside of minipage environments (for example, see Section 28 on page 104). When minipages are placed side-by-side, LaTeX places them such that their reference points are vertically aligned. By default, the minipage's reference point is vertically centered on its left edge. An optional argument modifies the location of a minipage's reference point.

`[b]` causes the minipage's reference point to be vertically aligned with the reference point of the bottom line in the minipage.

`[t]` causes the minipage's reference point to be vertically aligned with the reference point of the top line in the minipage.

Note the `[b]` does *not* put the reference point at the bottom of the minipage. Likewise, the `[t]` does *not* put the reference point at the top of the minipage.

When the minipage contains only one line, the `[b]` and `[t]` options produce the same results. For example, both

```
\begin{center}
   \begin{minipage}[b]{.25\linewidth}
      \centering
      \includegraphics[width=1in]{graphic}
   \end{minipage}%
   \begin{minipage}[b]{.25\linewidth}
      \centering
      \includegraphics[width=1in,angle=-45]{graphic}
   \end{minipage}
\end{center}
```

and

```
\begin{center}
   \begin{minipage}[t]{.25\linewidth}
      \centering
      \includegraphics[width=1in]{graphic}
   \end{minipage}%
   \begin{minipage}[t]{.25\linewidth}
      \centering
      \includegraphics[width=1in,angle=-45]{graphic}
   \end{minipage}
\end{center}
```

produce Figure 4. In both of these cases, reference point of the minipage is the reference point (original lower-left corner) of the EPS graphic.

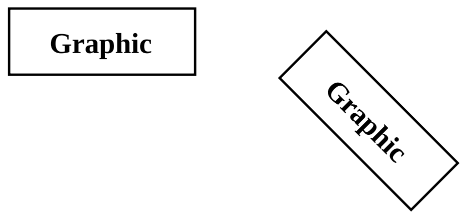


Figure 4: minipages with `[b]` or `[t]` options

11.4.1 Aligning the Bottoms of Minipages

One method for aligning the bottoms of minipages is to force the minipage baseline be the bottom of the minipage. Recall that the minipage `[b]` option makes the minipage baseline be the baseline of the minipage's bottom line.

If the bottom line of the minipage happens to have zero depth, then the last line's reference point is the bottom of the line and then the minipage `[b]` option would make the minipage baseline be the bottom of the minipage. Similarly, if a line with zero height and zero depth is added just before `\end{minipage}`, then the `[b]` option makes the minipage's baseline be the bottom of the minipage.

The command `\par\vspace{0pt}` creates such a zero-height, zero-depth line. Since the baseline of this zero-depth line is the bottom of the minipage, the `[b]` option now aligns the bottom of the minipage. For example

```
\begin{center}
   \begin{minipage}[b]{.25\linewidth}
      \centering
      \includegraphics[width=1in]{graphic}
      \par\vspace{0pt}
   \end{minipage}%
   \begin{minipage}[b]{.25\linewidth}
      \centering
      \includegraphics[width=1in,angle=-45]{graphic}
      \par\vspace{0pt}
   \end{minipage}
\end{center}
```

produces Figure 5.

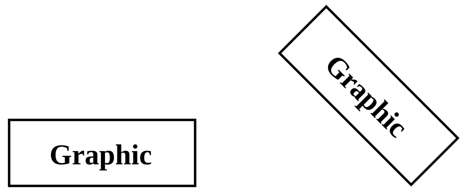


Figure 5: Minipages with Bottoms Aligned

11.4.2 Aligning the Tops of Minipages

When a zero-height, zero-depth line is added to the top of the minipage, the `[t]` option makes the minipage baseline be the top of the minipage. When this is done with multiple side-by-side minipages, the tops of the minipages are aligned.

The command `\vspace{0pt}` inserts a zero-height, zero-depth line at the top of the minipage. Since the baseline of this zero-height line is the top of the minipage, the `[t]` option now aligns the top of the minipage. For example

```
\begin{center}
   \begin{minipage}[t]{.25\linewidth}
      \vspace{0pt}
      \centering
      \includegraphics[width=1in]{graphic}
   \end{minipage}%
   \begin{minipage}[t]{.25\linewidth}
      \vspace{0pt}
      \centering
      \includegraphics[width=1in,angle=-45]{graphic}
   \end{minipage}
\end{center}
```

produces Figure 6.

This aligns the tops of the minipages with the current baseline. If it is instead desired to align the tops of the minipages with the top of the current line of text, replace `\vspace{0pt}` with `\vspace{-\baselineskip}`. This topic is mentioned in [3, pages 863-865].

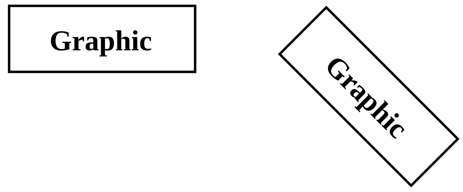


Figure 6: Minipages with Tops Aligned

12 Overlaying Two Imported Graphics

This section describes how to overlay two graphics. Note that there is no guarantee that the top graphic is transparent; it may have been created with an opaque background that hides the bottom graphic.

For example[18], the files `left.eps` and `right.eps` contain the graphics shown in Figure 7, then the commands

```
\makebox[0pt][l]{\includegraphics{left.eps}}%
\includegraphics{right.eps}
```

overlay the two graphics, as shown in Figure 8. The graphics are overlaid with their reference (lower-left) points coincident. In this particular example, the graphics had identical natural sizes so they overlaid perfectly without any scaling. Other pairs of graphics may require scaling (by `\includegraphics`, `\scalebox`, or `\resizebox`) to achieve the desired overlaying.

This overlaying code may seem mysterious unless one understands the `\makebox` command. The `\makebox[0pt][l]{...}` creates a zero-width box in which its argument is placed. When a width is specified (0 pt in this case), the typesetting algorithms allocate this much horizontal space regardless of the actual width of the contents. Thus, a left-justified zero-width box causes the LaTeX objects following the box to be typeset on top of the contents of the box.

(a) Contents of `left.eps`

(b) Contents of `right.eps`

Figure 7: Contents of Graphics Files

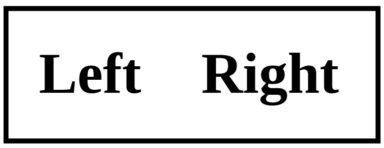


Figure 8: Two overlaid graphics

[18]Although this example overlays two EPS files, similar code can be used for overlaying other graphic formats.

12.1 Overpic Package

Another method for overlaying graphics is the overpic package, which defines a picture environment which is the size of the included graphic. See the overpic package documentation [27] for details.

13 Using Subdirectories

When importing a large number of graphics files, it may be desirable to store the graphics files in a subdirectory. For example, when the subdirectory is named `sub`, one may be tempted to then include the file `file.eps` with the following command

```
\includegraphics{sub/file.eps}
```

While this syntax works for most Unix and DOS TeX distributions, there are problems with such usage

Inefficiency

Whenever TeX opens a file, the filename is saved in TeX memory. When opening a large number of files, this lost memory may cause a poolsize error (see Section 13.4 on page 41). Since explicitly specifying subdirectories increases the filename length, it aggravates this pool space problem.

Unportability

One of LaTeX's advantages is that its files can be used on any platform. However, embedding the subdirectory in the filename results in the file becoming operating-system dependent. The file now cannot be used on VMS or Macintosh computers without significant modification.

Instead of embedding the subdirectory in the filename, there are two other options

1. The best method is to modify the TeX search path (see Section 13.1 on page 39).
2. Another method is to specify `sub/` in a `\graphicspath` command (see Section 13.3 on page 40). However, this is much less efficient than modifying the TeX search path.

Both of these options causing `\includegraphics` to automatically search the graphics subdirectory, allowing

```
\includegraphics{sub/file.eps}
```

to be replaced with

```
\includegraphics{file.eps}
```

13.1 TeX Search Path

Since the method for changing the directories in which TeX looks depends on the TeX distribution, it becomes very complicated to provide a general description. As an example, this section describe the strategy used by the web2c/teTeX Unix distributions. Although the method for changing the search path differs for other TeX distributions, most employ similar strategies.

For web2c/teTeX Unix distributions, the TeX search path can be modified by setting the `TEXINPUTS` environment variable. When using `csh` shells,

```
setenv TEXINPUTS /dir1:/dir2:
```

causes `/dir1` and `/dir2` to be searched *before* the default directories. Without the trailing colon, the default directories are not be searched. Setting `TEXINPUTS` with

```
setenv TEXINPUTS :/dir1:/dir2
```

causes `/dir1` and `/dir2` to be searched *after* the default directories, while

```
setenv TEXINPUTS /dir1::/dir2
```

causes `/dir1` to be searched *before* the default directories and `/dir2` to be searched *after* the default directories.

Putting `//` after a directory causes all of its subdirectories to be searched. For example,

```
setenv TEXINPUTS /dir1//:/dir2:
```

causes all the subdirectories (and sub-subdirectories) of `/dir1` to be searched. Be careful in using `//` as it may slow down the searching if the directory contains many files.

These examples also work for `sh` shells, although the syntax should be changed to

```
TEXINPUTS="/dir1:/dir2:"; export TEXINPUTS
```

When LaTeX finds files on the TeX path, it does not include the entire filename in the DVI file. As a result, old versions of `dvips` or `xdvi` which do not search the TeX path cannot find the file (see Section 14.4 on page 44).

13.2 Temporarily Changing the TeX Search Path

This section describes how a Unix shell script can temporarily change the TeX Search Path in order to find project-specific graphics files. Users can then construct a separate shell script for each of their projects, with each script specifying the directories that are unique to those projects.

For example, suppose a user is writing a journal paper and wants to create a unix shell script `latex_paper` that replaces the `latex` command. Create a file named `latex_paper` on the Unix search path containing

```
#!/bin/sh
TEXINPUTS= ~/PAPER/SUB1/:~/PAPER/SUB2/:$TEXINPUTS latex $@
```

Make the file executable with

```
chmod u+x latex_paper
```

Once this is done, typing

```
latex_paper file.tex
```

adds `~/PAPER/SUB1/:~/PAPER/SUB2/` to the beginning of `TEXINPUTS` before `latex file.tex` is run, allowing LaTeX to find any graphics stored in the `~/PAPER/SUB1/` or `~/PAPER/SUB2/` subdirectories.

A similar script called `dvips_paper` would also need to be written in order for `dvips` to find the graphics during DVI-to-PS conversion.

13.3 Graphics Search Path

By default, LaTeX looks for graphics files in any directory on the TeX search path. In addition to these directories, LaTeX also looks in any directories specified in the `\graphicspath` command. For example,

```
\graphicspath{{dir1/}{dir2/}}
```

tells LaTeX to also look for graphics files in `dir1/` and `dir2/`. For Macintosh, this becomes

```
\graphicspath{{dir1:}{dir2:}}
```

It is important to note that the file-searching associated with `\graphicspath` directories is much slower than that associated with `TEXINPUTS` directories. Furthermore, each file search done in a `\graphicspath` directory consumes additional pool space (see Section 13.4 on page 41).

Due to these inefficiencies, use of `\graphicspath` is generally discouraged. Instead, it is usually better to specify subdirectories by modifying the TEX search path (see Section 13.1 on page 39).

13.4 Conserving Pool Space

TEX reserves a portion of its memory called *pool space* for its internal passing of strings. Whenever TEX opens a file (or tries to open a file), some pool space is permanently used. When opening a large number of files, this lost memory may cause TEX to run out of pool space, causing an error similar to

```
! TeX capacity exceeded, sorry [poolsize=72288]
```

Since the amount of lost pool space is a function of the length of the filename, specifying subdirectories aggravates this pool space problem.

With the exception of the latest web2c version and some commercial distributions, the only way to increase the pool size is to recompile TEX. Fortunately, the following pool-conservation rules usually solve the problem.

- Avoid excessively-long file names.
- Don't include the subdirectory names

  ```
  \includegraphics{sub/file.eps}
  ```

  Instead, change the TEX search path or move the files out of the subdirectory.
- Don't use the `\graphicspath` command.

  ```
  \graphicspath{{dir1/}{dir2/}}
  ...
  \includegraphics{file.eps}
  ```

  causes `\includegraphics` to try to open the following files

  ```
  file.eps
  dir1/file.eps
  dir2/file.eps
  ```

  Each of these attempts consumes pool space. Instead of using `\graphicspath`, modify the TEX search path.
- Specify the entire filename, do not omit the files extension/suffix (e.g., `.eps`). With the default `\DeclareGraphicsExtensions` (see Section 9.1 on page 29), the command

  ```
  \includegraphics{file}
  ```

  causes `\includegraphics` to try to open the following files

  ```
  file.eps
  file.ps
  file.eps.gz
  file.ps.gz
  file.eps.Z
  ```

  This is especially inefficient when used in conjunction with `\graphicspath`.

  Issuing a `\DeclareGraphicsExtensions` command with a minimal number of extensions minimizes the inefficiency of omitting the extension.

Note that `\includegraphics` only consumes pool space when it opens or attempts to open a file. Since `\includegraphics` opens files to determine the graphic's BoundingBox, an effective but inconvenient method of preventing pool space consumption is to specify the BoundingBox parameters with the `\includegraphics` command's `bb` option (see Table 1 on page 25).

14 Compressed and Non-EPS Graphics Files in dvips

As described in Section 1, for `dvips`-style documents, LaTeX shifts the graphics-insertion burden onto the DVI programs. This means that a LaTeX document can use any graphic format which is supported by the DVI program.

While virtually all DVI-to-PS converters support insertion of EPS graphics, few converters support non-EPS graphics. This means the using non-EPS graphics with a `dvips`-style document generally requires converting the non-EPS graphics into EPS form. This can be done in two ways:

Convert the ahead of time Before DVI-to-PS conversion, use a graphics-conversion program to convert the non-EPS graphic into EPS format. This converted EPS file is stored and subsequently used by the DVI-to-PS converter.

Convert on-the-fly During the DVI-to-PS conversion, the DVI-to-PS converter calls a graphics-conversion program, with the graphics-conversion output being piped back into DVI-to-PS converter and inserted into the final PS file.

The disadvantage of converting ahead of time is that it requires storage of a second version of the graphics file. Although the "Convert on-the-fly" procedure does not require the additional storage, the same graphics-conversion computations are repeated at every DVI-to-PS execution. This is a storage-vs-speed tradeoff, but most users prefer the speed of the "Convert ahead of time".

Rather than directly incorporating graphics-conversion routines, `dvips` provides a mechanism for calling external conversion programs[19]. This mechanism can be accessed from LaTeX by use `command` argument of `\DeclareGraphicsRule`. This is more flexible than direct support because it keeps the graphics-conversion uncoupled from the DVI-to-PS conversion, allowing users to use the graphics-conversion program of their choice.

When using `dvips` and an operating system which supports pipes[20] one can use `\DeclareGraphicsRule` (see Section 9.2 on page 30) to specify an operation to be performed on the file. Making this operation a decompression command allows compressed graphics files to be used. Making this operation a graphics-conversion command allows non-EPS graphics files can be used.

Since `dvips` is currently the only DVI-to-PS converter with this capability, everything in this section requires `dvips`. Users need to pass the `dvips` option to the `graphicx` package. This can be done by either specifying the `dvips` global option in the `\documentclass` command

```
\documentclass[dvips,11pt]{article}
```

or by specifying the `dvips` option in the `\usepackage` command

```
\usepackage[dvips]{graphicx}
```

Specifying the `dvips` option in `\documentclass` it is preferred because it passes the `dvips` option to all packages.

---

[19]This requires an operating system which supports pipes. For example, Unix supports pipes while DOS does not.

[20]For example, Unix supports pipes while DOS does not.

14.1 Compressed EPS Example

The steps for using compressed EPS files are

1. Create an EPS file (`file1.eps` for example)
2. Store the BoundingBox line in another file (`file1.eps.bb`)
3. Compress the EPS file. For example, on many platforms, the command

   ```
   gzip -9 file1.eps
   ```

   creates the compressed file `file1.eps.gz`. The `-9` (or `-best`) option specifies maximum compression.
4. Include the proper `\DeclareGraphicsRule` command before the `\includegraphics` command in the LaTeX file. The `\DeclareGraphicsRule` command informs LaTeX how to treat the particular suffix (see Section 9.2 on page 30). For example

   ```
   \documentclass[dvips]{article}
   \usepackage{graphicx}
   \begin{document}
      \DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}{‘gunzip -c #1}
      \begin{figure}
           \centering
           \includegraphics[width=3in]{file1.eps.gz}
           \caption{Compressed EPS Graphic}
           \label{fig:compressed:eps}
      \end{figure}
   \end{document}
   ```

In this particular case, the `\DeclareGraphicsRule` command is actually not necessary because it happens to be one of the graphics rules pre-defined in `dvips.def`. If another compression program or suffixes were used, the `\DeclareGraphicsRule` command would be mandatory. For example, if the BoundingBox file had been stored in `file1.bb`, the corresponding `\DeclareGraphicsRule` would be

```
\DeclareGraphicsRule{.eps.gz}{eps}{.bb}{‘gunzip -c #1}
```

14.2 Non-EPS Graphic Files

While it is easy to insert EPS graphics into LaTeX documents, it is not as straightforward to insert non-EPS graphics (GIF, TIFF, JPEG, PICT, etc.). A simple solution is to determine whether the application which generated the non-EPS graphic also generates EPS output. If not, a graphics-conversion program (see Section 6.2 on page 17) must be used to convert the graphics to PostScript.

Since a non-EPS graphics file may be smaller than the corresponding EPS file, it may be desirable to keep the graphics in a non-EPS format and convert them to PostScript when the DVI file is converted to PostScript. If `dvips` is used, this on-the-fly conversion can be specified by the command option in `\DeclareGraphicsRule`. For example, using on-the-fly conversion to insert `file2.gif` into a LaTeX document requires the following steps

1. Find a GIF-to-EPS conversion program (assume it's called `gif2eps`)
2. One needs to create a BoundingBox file which specifies the natural size of the `file2.gif` graphics. To do this, convert `file2.gif` to PostScript and
   (a) If the PostScript file contains a BoundingBox line, save the BoundingBox line in `file2.gif.bb`

(b) If the PostScript file contains no BoundingBox line, determine the appropriate BoundingBox (see Section 3.2 on page 12) and place those numbers in a `%%BoundingBox:` line in `file2.gif.bb`

3. Keep `file2.gif` and `file2.gif.bb` and delete the PostScript file.
4. Include `\DeclareGraphicsRule{.gif}{eps}{.gif.bb}{‘gif2eps #1}` before the `\includegraphics` command in the LaTeX file.

When `\includegraphics{file.gif}` is issued, LaTeX reads the BoundingBox from `file.gif.bb` and tells `dvips` to use `gif2eps` to convert `file.gif` to EPS.

14.3 GIF Example

While the commands necessary for including non-EPS graphics are dependent on the operating system and the graphics conversion program, this section provides examples for two common Unix conversion programs. The commands

```
\DeclareGraphicsRule{.gif}{eps}{.gif.bb}{‘convert #1 ’eps:-’ }
\begin{figure}
   \centering
   \includegraphics[width=3in]{file2.gif}
   \caption{GIF Graphic}
\end{figure}
```

use the `convert` program (part of the ImageMagick package) package to translate the GIF file into EPS. The command

```
convert file2.gif ’eps:-’
```

translates `file2.gif` into EPS format (specified by the “`eps:`” option), sending the result to standard output (specified by the “`-`” specification).

Alternatively, one can use the `ppm` utilities in which `giftoppm`, `ppmtopgm`, and `pgmtops` convert the GIF file to EPS via the `ppm` and grayscale `pgm` formats. In Unix, the piping between these programs is specified by the following `\DeclareGraphicsRule` command

```
\DeclareGraphicsRule{.gif}{eps}{.gif.bb}{‘giftoppm #1 | ppmtopgm | pgmtops}
```

14.4 TeX Search Path and dvips

When LaTeX encounters an `\includegraphics` command, it looks in the current directory for the file. If it does not find the file in the current directory, it searches through the TeX path for the file. When the DVI file is converted to PostScript, `dvips` performs the same search routine and everything works well. However, when an on-the-fly command is specified in the `\DeclareGraphicsRule` command, the on-the-fly command prevents `dvips` from properly searching the TeX path.

For example, the rule

```
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}{‘gunzip -c #1}
```

specifies that the `gunzip -c` command be used on files having a `.eps.gz` suffix. Suppose the following command is used

```
\includegraphics{file.eps.gz}
```

If `file.eps.gz` and `file.eps.bb` are in the current directory, the path-searching is not needed and everything works well. LaTeX uses `file.eps.bb` and `dvips` executes `gunzip -c file.eps.gz` to uncompress the file.

However, things don’t work if `file.eps.gz` and `file.eps.bb` are not in the current directory. If they are instead in the directory `/a/b/c/` (on the TeX path), LaTeX searches the path to find `/a/b/c/file.eps.bb`. However, problems occur when dvips executes ‘gunzip -c file.eps.gz because gunzip cannot find file.eps.gz. If the TEX distribution uses a recent kpathsea library (as does the teTeX distribution), this problem can be solved by the following graphics rule

```
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}%
                    {‘gunzip -c ‘kpsewhich -n latex tex #1‘}
```

which uses kpsewhich to find the file for gunzip. The

```
‘kpsewhich -n latex tex #1
```

command causes dvips look for the compressed file on the TEX search path. The full filename (including subdirectories) is then appended to the gunzip -c command, allowing gunzip to find the file even though it is not in the current directory.

While this new \DeclareGraphicsRule command can be placed at the beginning of every document, it may be more convenient to add the following to the graphics.cfg file

```
\AtEndOfPackage{%
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}%
                    {‘gunzip -c ‘kpsewhich -n latex tex #1‘}}
```

and leaving the existing \ExecuteOptions{dvips} line.

15 The PSfrag Package

While there are many drawing and analysis packages which produce EPS files, most do not support symbols and equations as well as LATEX. The PSfrag package allows LATEX users to replace text strings in EPS files with LATEX text or equations.

Table 5: PSfrag Options

PStext	Text in EPS file to be replaced.
posn	*(Optional, Defaults to [Bl].)* Position of placement point relative to new LATEX text.
PSposn	*(Optional, Defaults to [Bl].)* Position of placement point relative to existing EPS text.
scale	*(Optional, defaults to 1.)* Scaling factor for the text. For best results, avoid using the scaling factor and instead use LATEX type-size commands such as \small and \large
rot	*(Optional, defaults to zero.)* When an angle is specified, it is the angle of rotation of the new text relative to the existing text. The angle is in degrees with a counterclockwise rotation being positive. This option is especially useful when dealing with applications which only allow horizontal text in their EPS files.
text	The LATEX text to insert into the EPS graphic. Like regular LATEX text, math formulas must be enclosed by dollar signs (e.g., `$\frac{1}{2}$` or `$x^2$`).

PSfrag 3.0 was totally re-written and released in November 1996. Previous versions of PSfrag required running a preprocessor (such as ps2frag or ps2psfrag) to identify and tag all the text in the EPS file. Since PSfrag 3.0 requires no preprocessing, it does not require any external programs such as perl or ghostscript. PSfrag 3.0 only requires a recent LATEX and the graphics bundle distributed with LATEX. Reference [29] provides complete documentation on PSfrag 3.0.

An additional benefit of PSfrag rewrite is that it now supports compressed EPS graphics. However, the `\tex` command (described in Section 15.3 on page 48) cannot be used to embed LaTeX text in compressed graphics.

To use PSfrag, create an EPS file and then perform the following steps

1. Include `\usepackage{psfrag}` in the preamble of the LaTeX document.
2. In the document, use the `\psfrag` command to specify the EPS text to replace and the LaTeX string to replace it. This makes the specified substitution occur in any subsequent `\includegraphics` command issued in the same environment.
3. Use the `\includegraphics` command as usual.

The LaTeX `\psfrag` command has the following syntax

```
\psfrag{PStext}[posn][PSposn][scale][rot]{text}
```

with its arguments described in Table 5.

The `posn` and `PSposn` options are one of the 12 points (such as `[tl]`, `[br]`, `[cc]`) shown in Figure 3 on page 28. If the optional arguments are not issued, the point defaults to `[Bl]`. Any missing letters default to `c` (e.g., `[]` and `[c]` are equivalent to `[cc]`, `[l]` is equivalent to `[lc]`). See [29] for examples of various combinations of placement points.

Note that `\psfrag` matches *entire* text strings. Thus the command

```
\psfrag{pi}{$\pi$}
```

replaces the string `pi` with π, but does not affect the strings `pi/2` or `2pi`. Separate `\psfrag` commands must entered for these strings.

Problems with Kerning

PSfrag cannot perform the replacement unless the entire EPS string is constructed with a single PostScript command. Some programs break string up into sub-strings or individual letters in order to perform kerning. For example, Corel Draw produced the following EPS code to place the string "Hello World"

```
0 0 (Hello W) @t
1080 0 (orld) @t
```

Since PSfrag sees this as two unrelated strings "Hello W" and "orld", it cannot perform any replacement of "Hello World". If the kerning cannot be manually turned off, using Courier or other monospaced fonts often prevents the kerning[21]. If the kerning cannot be avoided, only single-character replacement strings can be used.

15.1 PSfrag Example #1

The commands

```
\includegraphics{pend.eps}
```

include the graphic without any PSfrag replacement, producing Figure 9. The commands

```
\psfrag{q1}{$\theta_1$}
\psfrag{q2}{$\theta_2$}
\psfrag{L1}{$L_1$}
\psfrag{L2}{$L_2$}
\psfrag{P1}[][]{$P_1$}
\psfrag{P2}[][]{\large $P_2$}
\includegraphics{pend.eps}
```

[21] In order to avoid kerning, the font may need to be set to Courier *before* the text is created. That is, creating the text and then converting it to Courier may still result in kerning.

include the graphic with PSfrag replacement, producing Figure 10. The first four `\psfrag` commands position the new LaTeX text such that its left baseline point corresponds to the left baseline point of the EPS text. The last two `\psfrag` commands use the `[][]` options to position the LaTeX text such that its center corresponds to the center of the EPS text. Note that the `N` tag in Figure 10 is left unchanged, showing that not all EPS text has to be replaced.

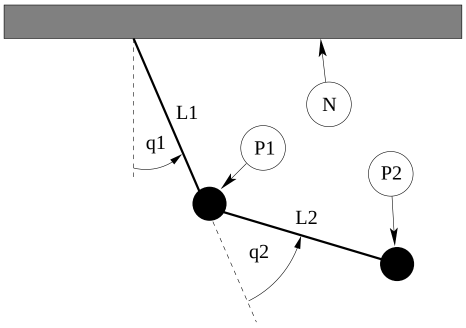


Figure 9: Without PSfrag Replacement

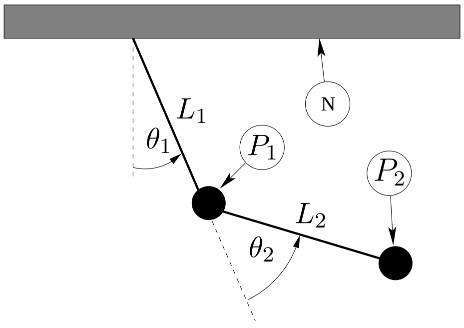


Figure 10: With PSfrag Replacement

15.2 PSfrag Example #2

This example demonstrates how the `\shortstack`, `\colorbox`, and `\fcolorbox` commands can be used with `\psfrag`.

shortstack The `\shortstack` command allows text to be stacked vertically, which can be used to substitute multiple lines of text for a single line of text. The lines of text are separated by the `\\` command.

colorbox The `\colorbox` command (part of the color package, which is distributed with LaTeX) places a rectangular color background behind an object. The distance that the background extends beyond the object is controlled by the `\fboxsep` length. For example,

```
\colorbox{white}{text}
```

places a rectangular white background behind `text`. See the graphics bundle documentation [7] for details on `\colorbox`.

With PSfrag, `\colorbox` is useful for placing text at a location where lines or shading would make it difficult to view the text. Placing a white background behind the text prevents the drawing from obstructing the text.

fcolorbox The `\fcolorbox` command (also part of the color package) is similar to the `\colorbox` command, except that a frame is drawn around the background. The command `\fcolorbox{black}{white}{text}` puts a white background with a rectangular black frame behind `text`.

The thickness of the frame is controlled by the length `\fboxrule` and the spacing between the frame and the text/object is controlled by the length `\fboxsep`.

Figure 11 shows the graphic without PSfrag substitution. The commands

```
\psfrag{q1}[][]{\colorbox{white}{$q_1$}}
\psfrag{base}{\fcolorbox{black}{white}{Base}}
\psfrag{Actuator}[l][l]{\shortstack{Hydraulic\\ Actuator}}
\includegraphics{mass.eps}
```

use PSfrag to produce the graphics in Figure 12.

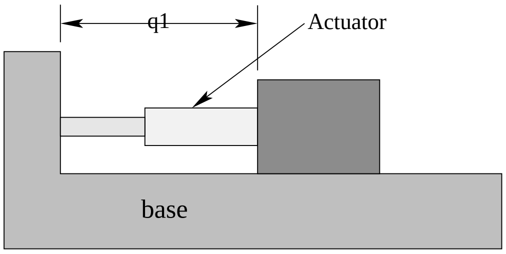


Figure 11: Without PSfrag Replacement

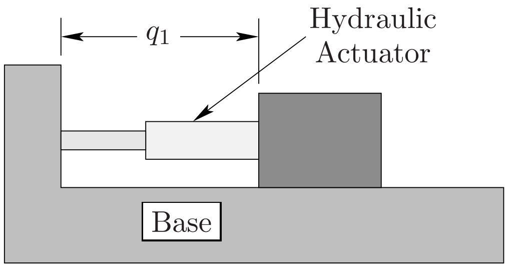


Figure 12: With PSfrag Replacement

15.3 LaTeX Text in EPS File

The recommended and most popular method for using PSfrag is the `\psfrag` command described in the previous section. An alternative, less efficient, method for using PSfrag is the `\tex` command, which embeds the LaTeX text directly in the EPS file. See [29] for more information.

15.4 Figure and Text Scaling with PSfrag

If a graphic using PSfrag is scaled, the PSfrag text is scaled along with the graphic. As a result, a subtlety of the graphicx package affects the size of the text.

- When the `width`, `height`, or `totalheight` options are used to size the graphic

  ```
  \includegraphics[width=3in]{file.eps}
  ```

  the PSfrag text is inserted *after* the scaling. Conversely,

  ```
  \resizebox{3in}{!}{\includegraphics{file.eps}}
  ```

  Includes the graphic at its natural size, inserts the PSfrag text, and then scales both the graphics and the text.

- Similarly, when scaling options are specified *before* rotation

  ```
  \includegraphics[width=3in,angle=30]{file.eps}
  ```

  the scaling is implicitly handled by the graphics inclusion function. However, when scaling options are specified *after* rotation

  ```
  \includegraphics[angle=30,width=3in]{file.eps}
  ```

  the graphic is first included at its natural size, then rotated, and then scaled. Since PSfrag replaces the new text during the graphics inclusion, the second command scales the new PSfrag text while the first command does *not*. When the included size of the EPS graphic greatly differs from its natural size, the two commands produce very different results.

See [29] for more information on the scaling of PSfrag text.

15.5 PSfrag and PDFTeX

PSfrag cannot be used with pdfTeX. If PSfrag substitution is needed, one option is to use the LaTeX-to-DVI-to-PostScript-to-PDF route that was used before pdfTeX. While this allows PSfrag substitution, users lose the advantages that pdfTeX provides.

A better (although more laborious) method is to use PSfrag indirectly with pdfTeX. This allows PSfrag substitution while also keeping the advantages of pdfTeX.

1. For each graphic that uses PSfrag, create a separate LaTeX file containing the PSfrag commands and the `\includegraphics` command. You must use `\pagestyle{empty}` to prevent page numbers from being placed on the page.

   Assume these PSfrag LaTeX files are named

   ```
   GraphicFrag00.tex
   GraphicFrag01.tex
   ...
   ```

2. At the operating system command line, perform the following steps

   ```
   latex  GraphicFrag00.tex
   dvips -E GraphicFrag00
   epstool --copy --bbox GraphicFrag00.ps GraphicFrag00.eps
   epstopdf GraphicFrag00.eps
   ```

   The first command creates `GraphicFrag00.dvi`. The second command creates `GraphicFrag00.ps`. The third command calculates the BoundingBox for `GraphicFrag00.ps` and inserts the BoundingBox and contents of `GraphicFrag00.ps` into `GraphicFrag00.eps`. The last command converts `GraphicFrag00.eps` into PDF format.

3. Repeat step 2 for `GraphicFrag01.tex`, ...
4. Use `\includegraphics` to include the resulting PDF files

   ```
   GraphicFrag00.pdf
   GraphicFrag01.pdf
   ...
   ```

   into the original LaTeX file.
5. Process the LaTeX file with `pdflatex`.

16 Including An EPS File Multiple Times

When the same EPS graphic is inserted multiple times, its EPS code appears multiple times in the final PS file. In particular, this often happens when a logo or other graphics are inserted into a document's header or footer. This section describes improved methods for inserting a graphic multiple times[22].

There are four common methods for including the same EPS graphics many times

1. Use `\includegraphics{file.eps}` wherever you want the graphic. This has two problems
   (a) LaTeX must find and read the file every time `\includegraphics` is used.
   (b) The EPS graphics commands are repeated in the final PS file, producing a large file.
2. Save the graphics in a LaTeX box and use the box wherever you want the graphic. This saves LaTeX time since it must only find and read the file once. However, it does not reduce the size of the final PostScript file.

   At the beginning of the file, include the following commands

   ```
   \newsavebox{\mygraphic}
   \sbox{\mygraphic}{\includegraphics{file.eps}}
   ```

[22] Although this document does not yet document it, users are encouraged to consider the graphicx-psmin package[20], which provides a way to include a repeated PostScript graphic only once in a PostScript document. This package only works when the DVI file is post-processed with `dvips` version 5.95b or later.

Then use the command `\usebox{\mygraphic}` wherever you want the graphic. (The graphics can be scaled by placing the `\usebox` command inside a `\scalebox` or `\resizebox` command.)

3. When the EPS file contains vector graphics (as opposed to bitmapped graphics), it is possible to write a PostScript command which draws the graphics[23]. The graphic can then be included by issuing the PostScript command wherever the graphic is needed. Section 16.1 on page 50 describes this procedure.

   Since the final PostScript file includes the graphics commands only once, the final PostScript file is much smaller. Note that since the graphics commands are stored in printer memory while the final PostScript file is being printed, this method *may* cause the printer to run out of memory and not print the document.

   Although this method results in a small final PostScript file, it still requires LaTeX to find and read the file containing the PostScript commands.

4. Like the previous method, define a PostScript command which draws the graphics, but include this command in a LaTeX box. This results in a small final PostScript file and only requires LaTeX to find and read the file once.

16.1 Defining a PostScript Command

This section describes how to created a PostScript command which draws the graphics from an EPS file containing vector graphics. This procedure does not work if the EPS file contains *bitmapped* graphics.

To convert the EPS graphics into a PostScript command, the EPS file must be broken into two files, one which defines the PostScript dictionary and the graphics commands, and another which includes the header information and the uses the previously-defined PostScript command. For example, an EPS file created by `Xfig` has the form

```
%!PS-Adobe-2.0 EPSF-2.0
%%Title: /tmp/xfig-fig017255
%%Creator: fig2dev Version 2.1.8 Patchlevel 0
%%CreationDate: Sun Sep  3 15:36:01 1995
%%Orientation: Portrait
%%BoundingBox: 0 0 369 255
%%Pages: 0
%%EndComments
/$F2psDict 200 dict def
$F2psDict begin
...
%%EndProlog

$F2psBegin
...
$F2psEnd
```

[23] While it is possible to construct a PostScript command which draws vector graphics, it turns out to be a "feature" of EPS that is impossible to construct such a command for bitmapped graphics. Bitmapped graphics are usually converted to EPS by having the `image` (or `colorimage`) PostScript operators read the current file as data. It is not possible to put such constructs into a PostScript procedure. It is possible to change the EPS file so that it passes the image data as PostScript strings rather than reading the file, but this is difficult to automate and generally requires a fair amount of hand editing of the PostScript.

Since most people are not PostScript experts, hand-editing the PostScript is generally not an option. If the graphic can be described by PostScript vector primitives, it may be possible to use the KVEC program (see page 19) to successfully convert the graphic to vector format.

Where ... indicates unlisted commands. The EPS file generally contains three parts

1. The header commands which begin with `%`
2. The Prolog section which starts with

   ```
   /$F2psDict 200 dict def
   ```

   and ends with

   ```
   %%EndProlog
   ```

   The Prolog defines the commands in the PostScript dictionary used by the EPS file. In this example, the dictionary is named `$F2psDict` although other names can be used.
3. The last part contains the commands used to draw the graphics.

Suppose the above EPS file is named `file.eps`. Create the files `file.h` and `file.ps` where `file.h` contains

```
/$F2psDict 200 dict def
$F2psDict begin
...
%%EndProlog

/MyFigure {
$F2psBegin
...
$F2psEnd
} def
```

and `file.ps` contains

```
%!PS-Adobe-2.0 EPSF-2.0
%%Title: /tmp/xfig-fig017255
%%Creator: fig2dev Version 2.1.8 Patchlevel 0
%%CreationDate: Sun Sep  3 15:36:01 1995
%%Orientation: Portrait
%%BoundingBox: 0 0 369 255
%%Pages: 0
%%EndComments
$F2psDict begin MyFigure end
```

`file.h` defines the dictionary and defines the PostScript command `/MyFigure`, while `file.ps` contains the header information and uses the PostScript command defined in `file.h`. In particular, it is important that the `file.ps` header includes the `%!PS...` line and the BoundingBox line. The graphics can then be used in the LaTeX document as

```
\documentclass{article}
\usepackage{graphicx}
\special{header=file.h}
\begin{document}
   ...
   \includegraphics[width=2in]{file.ps}
   ...
   \includegraphics[totalheight=1in]{file.ps}
   ...
\end{document}
```

Note that the original file `file.eps` is not used. Since the graphics commands in `file.h` are only included once, the final PostScript file remains small. However, this still requires LaTeX to find and read `file.ps` whenever the graphics are used. The following commands save the graphics in a LaTeX box to produce a small final PostScript file while reading `file.ps` only once.

```
\documentclass{article}
\usepackage{graphicx}
\special{header=file.h}
```

```
\newsavebox{\mygraphic}
\sbox{\mygraphic}{\includegraphics[width=2in]{file.ps}}

\begin{document}
   ...
   \usebox{\mygraphic}
   ...
   \resizebox*{1in}{!}{\usebox{\mygraphic}}
   ...
\end{document}
```

Like the previous example, these commands produce a 2-inch wide graphic and another graphic whose totalheight is 1 inch.

16.2 Graphics in Page Header or Footer

An easy method of including graphics in the heading is to use the fancyhdr package (an improved version of the old fancyheadings package) which is documented by [17]. The header consists of three parts: its left field, its center field, and its right field. The `\fancyhead` command specifies the contents of the header fields, with the `L,C,R` options specifying which field(s) the command modifies. For example

```
\pagestyle{fancy}
\fancyhead[C]{My Paper}
```

causes the center header field to be "My Paper", while

```
\pagestyle{fancy}
\fancyhead[L,R]{\textbf{Confidential}}
```

causes both the left and right header fields to be "**Confidential**". If no `L,C,R` option is specified, it applies to all three header fields. Thus `\fancyhead{}` is used to clear all the header fields. The `\fancyfoot` command similarly specifies the left, center, and right footer fields.

Graphics in Page Header/Footer

The commands in the fancyhdr package can insert graphics in the headers and footers. For example, after splitting the EPS file `file.eps` into the two file `file.h` and `file.ps` as described in Section 16.1 on page 50, the commands

```
\documentclass{article}
\usepackage{fancyhdr,graphicx}

\renewcommand{\headheight}{0.6in}  %% must be large enough for graphic
\renewcommand{\textheight}{7.5in}

% Define PostScript graphics command
\special{header=file.h}

% Save graphics in LaTeX box
\newsavebox{\mygraphic}
\sbox{\mygraphic}{\includegraphics[totalheight=0.5in]{file.ps}}

\pagestyle{fancy}
\fancyhead{}   % clear all header fields
\fancyhead[L]{\usebox{\mygraphic}}
\fancyfoot{}   % clear all footer fields
\fancyfoot[C]{\thepage}
\renewcommand{\headrulewidth}{0.5pt}
\renewcommand{\footrulewidth}{0pt}

\begin{document}
...
\end{document}
```

places the graphics at the top left of each "fancy" page with a 0.5 pt horizontal line drawn under the header. Additionally, the page number is placed at the bottom center of each page, with no horizontal line drawn above the footer. Note that this does not affect "plain" pages.

Odd/Even Headings When the [twoside] documentclass option is used, one may want to individually specify the odd and even page headers/footers. The E,O \fancyhead options specify the even and odd page headers, respectively. If the E,O options are not specified, the command applies to both even and odd pages. Likewise the E,O \fancyfoot options specify the even and odd page footers. For example,

```
\pagestyle{fancy}
\fancyhead[LE]{My Paper}
\fancyhead[RO]{My Name}
\fancyfoot[C]{\thepage}
```

places "My Paper" in the upper left of even fancy pages, "My Name" in the upper right of odd fancy pages, and the page number in the bottom center of all fancy pages. Replacing the

```
\fancyhead[L]{\usebox{\mygraphic}}
```

command in the above example with

```
\fancyhead[LE,RO]{\usebox{\mygraphic}}
```

places the graphic at the top outside (the left side of even pages, right side of odd pages) of all fancy pages.

Modifying Plain Pages The \fancyhead commands only apply to pages whose style are "fancy". Even though \pagestyle{fancy} causes the document to have a fancy page style, some pages (title pages, table of contents pages, the first page of chapters, etc.) are still given a plain pagestyle by default.

The \fancypagestyle command can be used to modify the plain pagestyle. For example, adding the following code to the above example causes the graphic to also be placed at the upper left of plain pages.

```
\fancypagestyle{plain}{%
   \fancyhead{}   % clear all header fields
   \fancyhead[L]{\usebox{\mygraphic}}
   \fancyfoot{}   % clear all footer fields
   \fancyfoot[C]{\thepage}
   \renewcommand{\headrulewidth}{0.5pt}
   \renewcommand{\footrulewidth}{0pt}}
```

When the twoside documentclass option is used, replacing both of the

```
\fancyhead[L]{\usebox{\mygraphic}}
```

commands with

```
\fancyhead[LE,RO]{\usebox{\mygraphic}}
```

places the graphic at the top outside of every page (both plain and fancy).

16.3 Watermark Graphics in Background

In addition to adding graphics to the headers and footers, the fancyhdr package can place graphics behind in the text, which is useful for creating a logo or seal watermark.

The following example places the graphics in file.eps on every page (both fancy and plain).

```
\documentclass{article}
\usepackage{graphicx,fancyhdr}

%%% store graphics in a box
\newsavebox{\mygraphic}
\sbox{\mygraphic}{\includegraphics[keepaspectratio,
                 height=0.8\textheight,
                 width=0.8\linewidth]{file.eps}}

\pagestyle{fancy}
\fancyhead{}
\fancyhead[C]{\setlength{\unitlength}{1in}
          \begin{picture}(0,0)
            \put(-2.2,-6){\usebox{\mygraphic}}
          \end{picture}}

\fancypagestyle{plain}{%
     \fancyhead{}%
     \fancyhead[C]{\setlength{\unitlength}{1in}
                 \begin{picture}(0,0)
                   \put(-2.2,-6){\usebox{\mygraphic}}
                 \end{picture}}}

\begin{document}
  ...
\end{document}
```

The above example places the graphics such that their lower left corner is 2.2 inches to the left and 6 inches below the center of the header. The graphic position can be adjusted by changing these two numbers.

Since the header is typeset before the text, this example causes the text to appear on top of the graphics. Since the footer is typeset after the text, putting the graphics in the footer causes the graphics to appear on top of the text.

If the contents of `file.eps` contain vector (not bitmapped) graphics, a much smaller final PostScript file can be obtained by using the procedure described in Section 16.1 on page 50.

16.3.1 eso-pic Package

Another method for adding LaTeX object on every page is the eso-pic package which defines a zero-length picture environment with basepoint at the lower left corner of the page. See the eso-pic package documentation [15] for details.

Part IV
The Figure Environment

17 The Figure Environment

When using a word processor, figures appear exactly where the user places them[24]. Since these figures cannot be split, they often lead to poor page breaks that leave large chunks of blank space at the bottom of pages. To achieve a professional-looking document, the author must manually rearrange the figures to avoid these poor page breaks. This figure-shuffling becomes quite tedious, especially since it must be repeated whenever the document is modified.

LaTeX provides floating figures which automatically move to esthetically-pleasing locations, producing professional-looking documents without all of the figure-moving drudgery. However, these floating figures often bother new users who are accustomed to manual figure placement. Taking advantage of LaTeX's floating figures requires the following

Don't compose text which is dependent on figure placement.

Using the phrase "This figure..." or "The following figure..." requires the figure to be in a certain location. Using the phrase "Figure 14..." allows the figure to be positioned anywhere.

Relax.

Some users get quite worried when a figure isn't placed exactly where they want it. Figure placement is LaTeX's job; users generally should not worry about it.

Summary of Advice The following pages describe how the LaTeX determines float locations which obey typesetting rules for a professional-looking document. For convenience, the solutions to the most-common float-placement problems are listed below.

1. Don't handcuff LaTeX. The more float placement options are given to LaTeX, the better it handles float placement. In particular, the `[htbp]` and `[tbp]` options work well. See Section 17.2 on page 58.
2. Many people find the default float parameters are too restrictive. The following commands

```
\setcounter{topnumber}{4}
\setcounter{bottomnumber}{4}
\setcounter{totalnumber}{10}
\renewcommand{\textfraction}{0.15}
\renewcommand{\topfraction}{0.85}
\renewcommand{\bottomfraction}{0.70}
\renewcommand{\floatpagefraction}{0.66}
```

set the float parameters to more-permissive values. See Section 18 on page 61.
3. LaTeX allows figures to float to the top of the current page, thus appearing before the reference in the text. Users who do not like this behavior should use the flafter package [18]. Include `\usepackage{flafter}` at the beginning of the document; no other commands are necessary.

[24] Although many word processors do allow figures to move around text (or vice-versa), the vast majority of the users do not use such capability because of either poor implementation by the software author and/or ignorance/indifference by the document author.

4. To guarantee that a figure does not float past a certain point, use the placeins package and issue a `\FloatBarrier` command. See Section 17.3 on page 59.

   Warning, overuse of `\FloatBarrier` indicates that either the float-placement is being micro-managed or the float parameters are set incorrectly, neither of which are good.

17.1 Creating Floating Figures

Floating figures are created by putting commands in a `figure` environment. The contents of the figure environment always remain in one chunk, floating to produce good page breaks. The floating figures can be automatically numbered by using the `\caption` command. For example, the following commands put the graphic from `graph.eps` inside a floating figure

```
\begin{figure}
   \centering
   \includegraphics[totalheight=2in]{graph.eps}
   \caption{This is an inserted EPS graphic}
   \label{fig:graph}
\end{figure}

The graph in Figure~\ref{fig:graph} on Page~\pageref{fig:graph}...
```

Notes about figures

- The optional `\label` command, can be used with the `\ref`, and `\pageref` commands to reference the caption. See Section 17.1.1 for additional information on references. The `\label` command must be placed immediately *after* the `\caption` command. Putting the `\label` before the `\caption` causes the `\ref` command to reference the last reference-able object (which often is the section or previous figure).
- If the figure environment contains no `\caption` commands, it produces an unnumbered floating figure.
- If the figure environment contains multiple `\caption` commands, it produces multiple figures which float together. This is useful in constructing side-by-side graphics (see Section 28 on page 104) or complex arrangements such as in Section 31 on page 110.
- By default, the caption text is used as the caption and also in the list of figures. The caption has an optional argument which specifies the list-of-figure entry. For example,

  ```
  \caption[List Text]{Caption Text}
  ```

  causes "Caption Text" to appear in the caption, but "List Text" to appear in the list of figures. This is useful when long, descriptive captions are used.
- The figure environment can only be used in *outer paragraph mode*, preventing it from being used inside any box (such as parbox or minipage).
- Figure environments inside of paragraphs

  ```
  ....text text text text text text
  \begin{figure}
     ....
  \end{figure}
  text text text text text text...
  ```

  are not processed until the end of the paragraph.

17.1.1 Defining a Reference Command

Instead of typing

```
Figure~\ref{fig:graph} on Page~\pageref{fig:graph}
```

it is more convenient to define the following command in the document preamble

```
\newcommand\Figpage[1]{Figure~\ref*{#1} on Page~\pageref*{#1}}
```

which allows the reference code to shortened to

```
\Figpage{fig:graph}
```

Conditional References

The above `\Figpage` definition always prints both the Figure's number and page number. In cases where the figure appears on the same page as the reference, it may be desirable to omit the page number. This can achieved by the following code

```
\newcommand\FigDiff[1]{Figure~\ref*{#1} on Page~\pageref*{#1}}
\newcommand\FigSame[1]{Figure~\ref*{#1}}
\newcommand\Figref[1]{\ifthenelse{\value{page}=\pageref{#1}}
                     {\FigSame{#1}}{\FigDiff{#1}}}
```

If the reference and figure are on the same page, the `\Figref` command calls the `\FigSame` command which displays "Figure 17". If the reference and figure are on different pages, the `\Figref` command calls the `\FigDiff` command which displays "Figure 17 on Page 25".

The varioref package [34] provides addition commands like this for referencing Figures, Tables, Section, etc.

17.1.2 hyperref Package

The hyperref package allows users to construct hyperlinks within LaTeX documents, most commonly in conjunction with `pdflatex`.

One feature of hyperref is that it redefines the `\ref` and `\cite` commands to be typeset as hyperlinks to the reference. The `\ref*` and `\cite*` commands are defined for references and cites without hyperlinks.

Because the hyperref package redefines many LaTeX commands, users should order their `\usepackage` to make hyperref be the last package loaded.

For more information, see the Hypertext Manual [21].

Hypertext References

When using the hyperref package, the `\ref` command typesets the figure number with a hyperlink to the figure. Since the figure number is relatively small, it may be difficult for readers to click on the actual figure number. To make clicking the hyperlink easier, the following code

```
\newcommand\Figlink[1]{\hyperref[#1]{Figure~\ref*{#1}} on Page~\pageref*{#1}}
```

defines a `\Figlink` command which turns the entire "Figure 17" reference into a single hyperlink.

17.2 Figure Placement

The `figure` environment has an optional argument which allows users to specify possible figure locations. The optional argument can contain any combination of the following letters

`h` *Here:* Place the figure in the text where the figure command is located. This option cannot be executed if there is not enough room remaining on the page.

`t` *Top:* Place the figure at the top of a page.

`b` *Bottom:* Place the figure at the bottom of a page[25].

`p` *Float Page:* Place the figure on a containing only floats.

Notes on figure placement:

- If no optional arguments are listed, the placement options default to `[tbp]`.

  The default arguments can be customized by redefining the internal command `\fps@figure`. For example, the following code

  ```
  \makeatletter
    \def\fps@figure{htbp}
  \makeatother
  ```

  causes the placement options to default to `[htbp]`.

- The order in which the placement options are specified does *not* make any difference, as the placement options are always attempted in the order `h-t-b-p`. Thus `[hb]` and `[bh]` are both attempted as `h-b`.

- The more float placement options are given to LaTeX, the better it handles float placement. In particular, the `[htbp]`, `[tbp]`, `[htp]`, `[tp]` options usually work well.

- Single-location options `[t]`, `[b]`, `[p]` `[h]` are problematic[26]. If the figure doesn't fit in the specified location, the figure becomes stuck, blocking the subsequent figures. A "Too Many Unprocessed Floats" error occurs if this log-jam of figures exceeds LaTeX's limit of 18 unprocessed floats (see Section 17.4 on page 60).

Also see Reference [1, pg 198].

When LaTeX "tries" to place a figure, it obeys the following rules

1. A figure can only be placed in the locations specified by its placement options.
2. The figure cannot cause the page to be overfull.
3. The float must be placed on the page where it occurs in the text, or on a later page[27]. Thus figures can "float later" but cannot "float earlier"
4. Figures must appear in order. Thus a figure cannot be placed until *all* previous figures are placed. Two ramifications of this rule are
   - A figure can never be placed "here" if there are any unprocessed figures.

[25] When a figure is placed at the bottom of a page, it is placed below any footnotes on the page. Although this may be objectionable, there currently is no way to change this arrangement.

[26] In fact, the `[h]` option should *never* be used. It is so bad that recent versions of LaTeX automatically change it to `[ht]`.

[27] Since a float can appear at the top of the page where it occurs in the text, it can appear before its occurrence in the text. If this is objectionable, the flafter package can be used to prevent this. No command is necessary to activate flafter; just include it in a `\usepackage` command.

- One "impossible-to-place" figure prevents any subsequent figure from being placed until the end of the document or until LaTeX's float limit is reached. See Section 17.4 on page 60.

Similarly, a table cannot be placed until *all* previous tables are placed. However, tables can leapfrog figures and vice-versa.

5. The aesthetic rules in Section 18 must be followed. For example, the number of floats on a page cannot exceed `totalnumber`. Specifying an exclamation point in the placement options (e.g., `\begin{figure}[!ht]`) makes LaTeX "try really hard" by ignoring the aesthetic rules which apply to text pages (`!` does not affect the aesthetic rules which apply to float pages).

17.3 Clearing Unprocessed Floats

A big advantage for using floats is that LaTeX is not required to place them immediately in the text. Instead, LaTeX can hold the float until it can place it at a better location. When a float has been read by LaTeX but not yet placed on the page, it is called a "unprocessed float." While the float-placing algorithm works well, sometimes it is necessary to force LaTeX to process any unprocessed floats.

Below are three methods for clearing processed floats. These commands should be used sparingly; their overuse is either a sign you are micro-managing your float placement or your float placement parameters have bad values (see Section 18 on page 61).

clearpage

The most basic method for clearing the unprocessed figures backlog is to issue a `\clearpage` command, which places all unprocessed floats and starts a new page. While this is effective, it is undesirable as it generally produces a partially-filled page.

FloatBarrier

For most situations, the best method for forcing float placement is the `\FloatBarrier` command provided by the placeins package. There are three ways of using placeins

1. The `\FloatBarrier` command causes all unprocessed floats to be processed immediately. Unlike `\clearpage`, it does not start a new page.
2. Since it is often desirable to keep floats in the section in which they were issued, the `section` option

   ```
   \usepackage[section]{placeins}
   ```

   redefines the `\section` command, inserting a `\FloatBarrier` command before each section.

   Note that this option is very strict. For example, if a new section start in the middle of a page, the `section` option does not allow a float from the old section to appear at the bottom of the page, since that is after the start of the new section.
3. The `below` option

   ```
   \usepackage[below]{placeins}
   ```

   is a less-restrictive version of the `section` option. It allows floats to be placed after the beginning of a new section, provided that some of the old section appears on the page.

The placeins package does not change the floats' float-placement options. For example, since \FloatBarrier can only place a [t] float at the top of a page, the \FloatBarrier command processes a [t] float by filling the remaining portion of the current page with whitespace and then placing the [t] float at the top of the next page. Similarly, since \FloatBarrier cannot place a [b] float "here", text is prevented from appearing below it.

Both of these examples once again demonstrate that LaTeX float placement is most effective when multiple float-placement options (such as [tbp] or [htbp]) are specified.

afterpage/clearpage

The afterpage package provides the \afterpage command which executes a command at the next naturally-occurring page break. Therefore, using

```
\afterpage{\clearpage}
```

causes all unprocessed floats to be cleared at the next page break.

Using \afterpage{\clearpage} command may not always solve float limit problems (see Section 17.4 on page 60). Since it puts does not execute the \clearpage until the end of the page, additional unprocessed floats may accumulate before the page break.

\afterpage{\clearpage} is especially useful when producing small floatpage figures. The \floatpagefraction (see Section 18.2 on page 61) prevents floatpage floats which are "too small" from being placed on a float page. Furthermore, since the ! float-placement modifier does not apply to float pages, [!p] does not override the \floatpagefraction restriction. Using \afterpage{\clearpage} is an easy method to override the \floatpagefraction restriction without causing a partially-filled text page.

17.4 Too Many Unprocessed Floats

If a float cannot be processed immediately, it is placed on the unprocessed float queue until it can be processed. Since, LaTeX only has room for 18 floats on this queue, having more than 18 unprocessed floats produces a "Too Many Unprocessed Floats" error. There are four possible causes for this error:

1. The most common problem is that the float placement options are incompatible with the float placement parameters. For example, a [t] figure whose height is larger than \topfraction becomes stuck. Since the other single-position options have similar problems, specify as many float placement options as possible.

2. Incompatible float fraction values may make it impossible to place certain floats. To avoid this, make sure any float fraction values satisfy the Section 18.2 guidelines.

3. In rare situations, users with many floats and many \marginpar notes (which use the same queue), may need a larger unprocessed float queue. Using the morefloats package increases the size of the unprocessed float queue from 18 to 36.

4. LaTeX's float placement queue is exceeded if more than 18 figures are specified without any text between them. Possible solutions include

(a) Scatter the figures in the text. This allows enough text to accumulate to force natural pagebreaks, making it easier for LaTeX to process the floats.
(b) Put `\clearpage` between some figures. This is inconvenient because it requires some iterations to avoid partially-full pages. Note that

```
\afterpage{\clearpage}
```

(which causes a `\clearpage` at the next naturally-occurring pagebreak) does not help in this situation because the float queue limit is reached before enough text is accumulated in order to trigger a pagebreak.
(c) Since there is no text, the figures don't need to float. Therefore, the best solution is probably to use the Section 21 procedure for constructing non-floating figures, separated by `\vspace` or `\vfill` commands to provide vertical spacing.

18 Customizing Float Placement

The following style parameters are used by LaTeX to prevent awkward-looking pages which contain too many floats or badly-placed floats. If these style parameters are changed anywhere in the document, they do not apply until the next page. However, if the parameters are changed in the document's preamble, they apply at the beginning of the document.

18.1 Float Placement Counters

The three counters in Table 6 prevent LaTeX from placing too many floats on a text page. These counters do not affect float pages. Specifying a `!` in the float placement options causes LaTeX to ignore theses parameters. The values of these counters are set with the `\setcounter` command. For example,

```
\setcounter{totalnumber}{2}
```

prevents more than two floats from being placed on any text page. Many peopler feel the default float placement counters are too restrictive and prefer larger values such as

```
\setcounter{topnumber}{4}
\setcounter{bottomnumber}{4}
\setcounter{totalnumber}{10}
```

Table 6: Float Placement Counters

`topnumber`	The maximum number of floats allowed at the top of a text page (the default is 2).
`bottomnumber`	The maximum number of floats allowed at the bottom of a text page (the default is 1).
`totalnumber`	The maximum number of floats allowed on any one text page (the default is 3).

18.2 Figure Fractions

The commands in Table 7 control what fraction of a page can be covered by floats (where "fraction" refers to the height of the floats divided by `\textheight`). The first three commands pertain only to text pages, while the last command pertains only to float pages. Specifying a `!` in the float placement options causes LaTeX to ignore the first three parameters, but `\floatpagefraction` is always used. The value of these fractions are set by `\renewcommand`. For example,

```
\renewcommand{\textfraction}{0.3}
```

lets floats cover no more than 70% of a text page.

Table 7: Figure Placement Fractions

`\textfraction`	The minimum fraction of a text page which must be occupied by text. The default is 0.2, which prevents floats from covering more than 80% of a text page.
`\topfraction`	The maximum fraction of a text page which can be occupied by floats at the top of the page. The default is 0.7, which prevents any float whose height is greater than 70% of `\textheight` from being placed at the top of a page. Similarly, if the combined height of multiple `t` floats is greater than 70% of `\textheight`, they all will not be placed at the top of a page, even if they number less than `topnumber`.
`\bottomfraction`	The maximum fraction of a text page which can be occupied by floats at the bottom of the page. The default is 0.3, which prevents any float whose height is greater than 30% of `\textheight` from being placed at the bottom of a text page.
`\floatpagefraction`	The minimum fraction of a float page that must be occupied by floats. Thus the fraction of blank space on a float page cannot be more than `1-\floatpagefraction`. The default is 0.5.

Placement Fraction Guidelines

The default placement fraction values prevent many and/or large floats from dominating text pages and also prevent small figures from being placed in a sea of whitespace on a float page. While the default values generally work well, sometimes they may be a bit too restrictive, resulting in figures floating too far from where they are issued. In these cases it may be desirable to set the placement fractions to more permissive values such as

```
\renewcommand{\textfraction}{0.15}
\renewcommand{\topfraction}{0.85}
\renewcommand{\bottomfraction}{0.70}
\renewcommand{\floatpagefraction}{0.66}
```

One must take care when adjusting placement fraction values, as unreasonable values can lead to poor formatting and/or “stuck” floats. To avoid such problems, the following guidelines should be used:

`\textfraction`

Setting `\textfraction` smaller than 0.15 is discouraged as it produces hard-to-read pages. If a figure’s height is more than 85% of `\textheight`, it almost certainly looks better by itself on a float page than squeezed on a text page with a couple of lines of text below it.

Furthermore, *never* set `\textfraction` to zero as permits a text page to have no text, which confuses LaTeX and leads to badly-formatted pages.

`\topfraction`

Never set `\topfraction` larger than `1 - \textfraction`, as that causes contradictions in the float-placing algorithm.

`\bottomfraction`

Since "good typesetting style" discourages large bottom figures, `\bottomfraction` is generally smaller than `\topfraction`. Never set `\bottomfraction` larger than 1 - `\textfraction`, as that causes contradictions in the float-placing algorithm.

`\floatpagefraction`

If `\floatpagefraction` is set very small, every float page contains exactly one float, resulting in excessive whitespace around small `p` figures.

If `\floatpagefraction` is larger than `\topfraction` , `[tp]` figures may become "stuck." For example, suppose the height of a `[tp]` figure is larger than `\topfraction` but smaller than `\floatpagefraction`, it becomes "stuck" because it is too large to be placed on a text page and too small to be placed on a float page. To prevent such stuck figures, `\floatpagefraction` and `\topfraction` should satisfy the following inequality:

$$\texttt{\textbackslash floatpagefraction} \leq \texttt{\textbackslash topfraction} - 0.05$$

The 0.05 term is due to the difference in the accounting of vertical space for text pages and float pages[28]. Likewise, if `[bp]` or `[hbp]` figures are used, `\floatpagefraction` and `\bottomfraction` should also satisfy

$$\texttt{\textbackslash floatpagefraction} \leq \texttt{\textbackslash bottomfraction} - 0.05$$

Note that the default values do not satisfy the second inequality, which may occasionally cause problems with `[bp]` and `[hbp]` figures.

18.3 Suppressing Floats

The `\suppressfloats` prevents additional floats from appearing at the top or the bottom of the current page. They do not affect figures with "here" placement or those with `!` in the placement options.

Putting `\suppressfloats[t]` immediately before a figure, prevents that float from appearing above the place where it appears in the text. The flafter package redefines LaTeX's float algorithm to prevent this for the entire document.

Table 8: Suppressfloats Options

`\suppressfloats[t]`	Prevents additional figures from appearing at the top of the current page.
`\suppressfloats[b]`	Prevents additional figures from appearing at the bottom of the current page.
`\suppressfloats`	Prevents additional figures from appearing at either the bottom or the top of the current page.

[28] Specifically, `\textfloatsep` and the other text-page float spacing *is* counted when comparing a figure with `\topfraction`, but the float page spacings are *not* counted in testing if a figure exceeds `\floatpagefraction`. As a result, `\textfloatsep` divided by `\textheight` (which is ≈ 0.05) should be subtracted from `\topfraction`. See Section 19.1 on page 64 for information on figure spacing.

19 Customizing the figure Environment

19.1 Figure Spacing

The lengths in Table 9 control how much vertical spacing is added between two figures or between a figure and text. Unlike most other LaTeX lengths, these three are rubber lengths, which provides spacing which can shrink or expand to provide better page formatting. These lengths are set with the `\setlength` command. For example,

```
\setlength{\floatsep}{10pt plus 3pt minus 2pt}
```

sets the "nominal" value of `\floatsep` to be 10 points. To improve page formatting, the float separation can be as small as 8 points or as large as 13 points.

Since LaTeX places `\intextsep` above and below each "here" float, two consecutive "here" floats are separated by two `\intextsep` spaces. This extra spacing can be avoiding by combining the two floats into a single float, although this may result in a less-attractive layout since it prevents the float-placement algorithm from placing the two floats separately.

The lengths listed in Table 9 do not affect the spacing of floats on float pages. The float-page spacing is controlled by the lengths listed in Table 10. The float-page spacings often use the `fil` unit to provide infinite stretchability, similar to the vertical space produced by `\vfill`. When multiple `fil` spaces appear in the same space, they expand proportionally to fill the space. For example, the default float-page parameters cause the space between float-page floats to be double the space above the top float or below the bottom float.

The `@` in the names of the Table 10 lengths mean they are internal commands[29]. As a result, any `\setlength` command which modifies the lengths must be surrounded by `\makeatletter` and `\makeatother`. For example,

[29] Any user code which accesses or redefines internal commands must be surrounded by `\makeatletter` and `\makeatother`.

Table 9: Figure Spacing for Text Pages

`\floatsep`	For floats at the top or bottom of a page, this is the vertical spacing between floats. The default is `12pt plus 2pt minus 2pt`
`\textfloatsep`	For floats at the top or bottom of a page, this is the vertical spacing between the float and the text. The default is `20pt plus 2pt minus 4pt`
`\intextsep`	For floats placed in the middle of a text page (i.e., with the `h` placement option), this is the vertical spacing above and below the float. The default is `12pt plus 2pt minus 2pt`

Table 10: Figure Spacing for Floatpages

`\@fptop`	This is the vertical spacing above the top floatpage float. The default is `0pt plus 1.0fil`
`\@fpsep`	This is the vertical spacing between floatpage floats. The default is `8pt plus 2.0fil`
`\@fpbot`	This is the vertical spacing below the bottom floatpage float. The default is `0pt plus 1.0fil`

```
\makeatletter
   \addtolength{\@fpsep}{4pt}
\makeatother
```

increases the space between floatpage floats by 4 points.

19.2 Horizontal Lines Above/Below Figure

Horizontal lines can be automatically drawn between the text and figures which appear at the top/bottom of the page by redefining the

```
\topfigurerule
\bottomfigurerule
```

commands. Although `\topfigrule` and `\bottomfigrule` are already defined as LaTeX commands, the strange way in which they are defined requires them to be redefined with `\newcommand` instead of `\renewcommand`.

To avoid disrupting the page formatting, these commands must have a zero height. Thus drawing 0.4 point line must be accompanied by a 0.4 point vertical backspace. For example,

```
\newcommand{\topfigrule}{\hrule\vspace{-0.4pt}}
```

Since `\topfigrule` is executed before the `\textfloatsep` spacing, the above command provides no spacing between the figure and the line. The following commands provide 5 points of space between the figure and the line.

```
\newcommand{\topfigrule}{%
   \vspace*{5pt}\hrule\vspace{-5.4pt}}
\newcommand{\botfigrule}{%
   \vspace*{-5.4pt}\hrule\vspace{5pt}}
```

The `\topfigrule` definition first moves 5 points down (into the `\textfloatsep` spacing) to provide space between the figure and the line. It then draws a 0.4 point horizontal line and moves back up 5.4 points to compensate for the previous downward motion. Likewise, the `\botfigrule` command draws a 0.4 point line with 5 points of spacing between the figure and the rule.

Since these commands place 5 points of space between the line and figure, the spacing between the line and the text is `\textfloatsep - 5pt` (see Section 19.1 on page 64).

The line thickness can be changed from the 0.4 point default by using the `\hrule` command's `height` option

```
\newcommand{\topfigrule}{%
   \vspace*{5pt}{\hrule height0.8pt}\vspace{-5.8pt}}
\newcommand{\botfigrule}{%
   \vspace*{-5.8pt}{\hrule height0.8pt}\vspace{5pt}}
```

Notes on figure rules:

- The `\topfigrule` and `\bottomfigrule` affect neither floatpage figures nor "here" figures (i.e., using the `h` option). If a "here" figure happens to be placed at the top or the bottom of the page, no line is drawn.

To implement its commands, LaTeX uses many internal commands which users generally do not need to access. To prevent these internal command names from accidentally conflicting with user-defined names, LaTeX includes a `@` in these internal command names. Since LaTeX command names can contain only letters, defining a command whose name contains `@` are normally not possible. However, when it is necessary for users to change the internal commands, the `\makeatletter` command causes LaTeX to treat `@` as a letter, thus allowing users to use `@` in command names. The `\makeatother` command causes LaTeX to revert to the normal behavior of treating `@` as a non-letter.

Table 11: Figure Rule Commands

`\topfigrule`	This command is executed after the last float at the top of a page, but before the `\textfloatsep` spacing (see Section 19.1 on page 64).
`\bottomfigrule`	This command is executed before the first float at the bottom of a page, but after the `\textfloatsep` spacing.

- The horizontal rules are as wide as the text, even if wider figures (see Section 23 on page 90) are used.
- The TeX `\hrule` command was used instead of LaTeX `\rule` command because the `\rule` would generate additional space when `\parskip` is not zero.

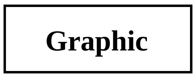
19.3 Caption Vertical Spacing

LaTeX assumes that captions are placed below the graphic, placing more vertical spacing above the caption than below it. As a result, the commands

```
\begin{figure}
   \centering
   \caption{Caption Above Graphic}
   \includegraphics[width=1in]{graphic}
\end{figure}
```

produce Figure 13, whose caption is placed quite close to the graphic.

Figure 13: Caption Above Graphic

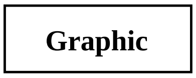


The caption spacing is controlled by the lengths `\abovecaptionskip` (which is `10pt` by default) and `\belowcaptionskip` (which is zero by default). The standard LaTeX commands `\setlength` and `\addtolength` are used to modify these lengths. For example, the commands

```
\begin{figure}
   \setlength{\abovecaptionskip}{0pt}
   \setlength{\belowcaptionskip}{10pt}
   \centering
   \caption{Caption Above Graphic}
   \includegraphics[width=1in]{graphic}
\end{figure}
```

produce Figure 14, which has has no extra space above the caption and 10 points of space between the caption and the graphic.

Figure 14: Caption Above Graphic



If a document has all its captions at the top of its floats, the commands

```
\setlength{\abovecaptionskip}{0pt}
\setlength{\belowcaptionskip}{10pt}
```

can be issued in the document's preamble to affect the caption spacing for *all* the document's captions (figures *and* tables). If a document contains captions at the top of some floats and at the bottom of other floats, it may be desirable to define the following command

```
\newcommand{\topcaption}{%
    \setlength{\abovecaptionskip}{0pt}%
    \setlength{\belowcaptionskip}{10pt}%
    \caption}
```

Then `\topcaption{caption text}` produces a caption which is properly spaced for the top of a float.

Two other methods for producing properly-spaced top captions are

- The caption package's `position=top` option in Table 16 on page 74 swaps the meaning of `\abovecaptionskip` and `\belowcaptionskip`
- The topcapt package [33], defines a `\topcaption` command which produces a caption with the `\abovecaptionskip` and `\belowcaptionskip` lengths interchanged.

19.4 Caption Label

By default, LaTeX inserts a caption label such as "Figure 13: " at the beginning of the the caption. The "Figure" portion can be changed by redefining the `\figurename` command. For example, the commands

```
\begin{figure}
   \centering
   \includegraphics[width=1in]{graphic}
   \renewcommand{\figurename}{Fig.}
   \caption{This is the Caption}
\end{figure}
```

produce Figure 15. The caption font, the ":" delimiter, and other caption characteristics can be customized with the caption package (see Section 20 on page 69).

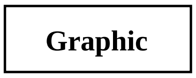


Fig. 15: This is the Caption

19.5 Caption Numbering

The default method for numbering the figures is Arabic (1, 2, 3, 4,...). This can be changed by redefining the `\thefigure` command.

The number of the current figure is stored in the `figure` counter. The `\thefigure` command specifies which of the counter numbering commands (`\arabic`, `\roman`, `\Roman`, `\alph`, `\Alph`) is used to print the counter value. For example,

```
\renewcommand{\thefigure}{\Roman{figure}}
```

causes the figures to be numbered with uppercase Roman numerals (I, II, III, IV,...).

Notes on figure numbering:

- There must be 26 or fewer figures to use the `\alph` or `\Alph` commands.
- Since Roman numbering produces longer figure numbers (e.g., XVIII vs. 18), using `\Roman` or `\roman` may cause spacing problems in the Table of Figures.

19.6 Moving Figures to End of Document

Some journals require that tables and figures be separated from the text. The endfloat package moves all the figures and table to the end of the document. Simply including the package

```
\usepackage{endfloat}
```

activates the package. The package supports many options which can be included in the `\usepackage` command, including

- Notes such as "[Figure 4 about here.]" are placed in approximately where the floats would have appeared in the text. Such notes can be turned off with the `nomarkers` package option

  ```
  \usepackage[nomarkers]{endfloat}
  ```

  The text of these notes can be changed by redefining the `\figureplace` and `\tableplace` commands. For example,

  ```
  \renewcommand{\figureplace}{%
    \begin{center}%
    [\figurename~\thepostfig\ would appear here.]%
    \end{center}}
  ```

  changes the `\figureplace` text.

- A list of figures is included before the figures and a list of tables is included before the tables. The `nofiglist` and `notablist` package options suppress these lists.

- The `fighead` and `tabhead` package options create section headers for the figures and tables, respectively.

- The figures appear before the tables. The `tablesfirst` package option reverses this order.

- A `\clearpage` command is executed after each figure and table, causing each float to appear on a page by itself. This can be changed by modifying the `\efloatseparator` command. For example,

  ```
  \renewcommand{\efloatseparator}{\mbox{}}
  ```

  places an empty `\mbox` after each float.

19.7 Adjusting Caption Linespacing

To doublespace a document, include either

```
\linespread{1.6}
```

or equivalently

```
\renewcommand{\baselinestretch}{1.6}
```

in the preamble[30]. In addition to doublespaced text, this also produces doublespaced captions and footnotes. To produce doublespaced text and singlespaced captions and footnotes, use the setspace package[31].

```
\usepackage{setspace}
\linestretch{1.5}
```

A 1.0 linestretch causes single-spaced text, a 1.25 linestretch causes space-and-a-half spaced text, and a 1.6 linestretch causes doublespaced text.

[30] Although it is generally considered poor style, these commands can also be used within a document to change the interline spacing. When these commands are used within a document, a fontsize command such as `\normalsize` must issued after the line-spacing command to put the new spacing into effect.

[31] Although the doublespace package also sets line spacing, it has not been properly updated to LaTeX 2_ε, causing it to interact with many packages. As a result, setspace should be used instead.

20 Customizing Captions with caption package

Section 19.4 on page 67 describes how to customize the caption label while Section 19.3 on page 66 describes how to customize the caption vertical spacing. The caption package[32] provides commands for customizing other caption characteristics. This section provides an overview of the caption package. Further details are found in the caption package documentation [12].

The caption package can be used with many types of floats as it officially supports the float, listings, longtable, rotating, sidecap, supertabular, and subfig packages. It also works with the floatfig, subfloat, and wrapfig packages.

ccaption **package** Although it is not described in this document, the ccaption (note the double-c) package also provides commands for customizing captions. It is described in [13].

20.1 Caption Package Overview

There are two aspects to the caption package

- The new variants of the `\caption` command which produce the captions are described in Section 20.2 and listed in Table 12.
- Section 20.3 on page 70 describes the two methods for specifying caption-customizing options. There are four types of options:

  Font Options customize the caption font[33]. These options are listed in Table 14 and Table 15 on page 73 with examples provided in Section 20.4.1 on page 76.

  Caption Spacing Options customize the caption vertical spacing. These options are listed in Table 16 on page 74 with examples provided in Section 20.4.2 on page 77.

  Caption Label Options customize the caption label and separator. These options are listed in Table 17 on page 74 with examples provided in Section 20.4.3 on page 79.

  Caption Formatting Options customize the caption formatting. These options are listed in Table 18 on page 75 with examples provided in Section 20.4.4 on page 80.

  Note that the tables listing the caption-package options (Tables 13 - 18) are grouped together on pages 73-75 to facilitate convenient reference. The examples are grouped together in Section 20.4 on page 76.
- Users can define a collection of caption options, called *caption styles*. The entire collection of options can be specified with the `style=` option. See Section 20.5.1 on page 84.
- Instead of just using the built-in option values, users can define their own option values as described in Section 20.5.2.

[32] Version 3 of the caption package replaces previous caption versions as well as the caption2 package.

[33] Although caption package provides commands to customize a caption's font, not every combination of font attributes necessarily exists in the font being used. For example, suppose the user specifies a font with roman family, small caps shape, and bold series. If that combination is not supported by the current font, then LaTeX may instead substitute a font with roman family, upright shape, and bold series.

20.2 Caption Commands

Section 17.1 on page 56 describes the `\caption` command and some customization is described in and Section 19. The `caption` package provides many more customization options.

The `caption` package slightly changes this `\caption` command and also introduces new variants as described in Table 12. The highlights include:

- The `caption` package changes the `\caption` command such that if the optional argument is specified but empty

  ```
  \caption[]{caption text}
  ```

  then no entry is made in the list of figures/tables for that caption.

- The new `\caption*` command displays the caption without a caption label and without entry in the list of tables.

- The new `\captionof` command allows a particular type of caption to be used anywhere: figure environment, table environment, or elsewhere in a document. For example

  ```
  \begin{figure}
      ....
      \captionof{table}[List of Tables Text]{Table Caption}
  \end{figure}
  ```

  produces a Table caption inside a figure environment. This is useful for

  1. Placing a table and figure side-by-side as described in Section 30 on page 109).
  2. Constructing marginal figures (see Section 22 on page 89).
  3. Constructing non-floating figures (see Section 21 on page 87).

  Note that the `\captionof` should always be used inside some type of environment (such as minipage) to avoid page breaks occurring between the caption and the float contents.

20.3 Customizing Captions with Caption Command

As mentioned earlier in Section 20.1 on page 69, the `caption` package allows the user to customize the caption font, spacing, label, and format. The options (listed in Tables 13 - 18) can be specified in either of two ways:

usepackage options

`\usepackage[options]{caption}` where `[options]` are any combination of options specified in Table 13. For example

```
\usepackage[margin=10pt,font=small,labelfont=bf]{caption}
```

causes all caption margins to be indented by an additional 10pt on both left and right sides, with the entire caption (label and text) having a small font size and the label having a bold font series.

captionsetup command

The command `\captionsetup{options}` causes specified options to be in effect for the remaining environment. (A `\captionsetup` command in a document's preamble effects the entire document.) For example

```
\captionsetup{margin=10pt,font=small,labelfont=bf}
```

Table 12: caption package caption Commands

Command	Description
`\caption{`⟨*caption text*⟩`}`	Uses ⟨*caption text*⟩ for both figure/table caption and for List of Figures/Tables. *(Same behavior as without* caption *package).*
`\caption[`⟨*list entry*⟩`]{`⟨*caption text*⟩`}`	Uses ⟨*caption text*⟩ for figure/table caption and ⟨*list text*⟩ for entry in List of Figures/Tables. *(Same behavior as without* caption *package).*
`\caption[ ]{`⟨*caption text*⟩`}`	Uses ⟨*caption text*⟩ for figure/table caption and creates *no* entry in List of Figures/Tables.
`\caption*{`⟨*caption text*⟩`}`	Uses ⟨*caption text*⟩ for figure/table caption but does not include caption label or separator (see Figure 16). No entry is generated for the List of Figures/Tables.
`\captionof{`⟨*float type*⟩`}[`⟨*list entry*⟩`]{`⟨*caption text*⟩`}`	If ⟨*float type*⟩ is `figure` then Figure caption and entry in List of Figures is generated, even if the `\captionof` command is located outside of a figure environment. Likewise, if ⟨*float type*⟩ is `table` then Figure caption and entry in List of Table is generated, even if the `\captionof` command is located outside of a table environment.
`\captionof*{`⟨*float type*⟩`}{`⟨*caption text*⟩`}`	Similar to `\captionof` command, the ⟨*float type*⟩ species whether a figure or table caption is generated. Similar to `\caption*` command, `\captionof*` command uses ⟨*caption text*⟩ for figure/table caption but does not include caption label or separator (see Figure 16). No entry is generated for the List of Figures/Tables.
`\ContinuedFloat`	Allows multiple `\caption` command to share same figure number. See Section 32.3 on page 114 and Section 33 on page 116.

causes all subsequent caption in the current environment to be indented by an additional 10pt on both left and right sides, with the entire caption (label and text) having a small font size and the label having a bold font series.

Table 13 describes the `\captionsetup` and `\clearcaptionsetup` commands.

The `\captionsetup` command has two advantages over specifying options in the `\usepackage` command

- The `\captionsetup` command has optional arguments that cause the options to apply to just figures or just tables.
- The `\captionsetup` can change the settings for an individual figure or table. For example:

```
\begin{figure}
   ...
   \captionsetup{justification=centering}
   \caption{This is the Caption Text}
\end{figure}
```

  causes the captions to be centered for this figure *only* but does not affect any other figure.

While the `\captionsetup` can be used to customize a single caption, this is generally considered bad style. In general, a user should issue `\captionsetup` commands the preamble and avoid using `\captionsetup` within the document.

Table 13: Caption Setup Commands from Caption Package

Command		Description
`\captionsetup[`⟨*float type*⟩`]{`⟨*options*⟩`}`		Set caption attributes
Examples	`\captionsetup{`⟨*options*⟩`}`	options apply to all captions
	`\captionsetup[figure]{`⟨*options*⟩`}`	options apply only to figure captions
	`\captionsetup[table]{`⟨*options*⟩`}`	options apply only to table captions
`\clearcaptionsetup{`⟨*float type*⟩`}`		changes caption attributes to defaults
Examples	`\clearcaptionsetup{figure}`	resets figure captions to have default options
	`\clearcaptionsetup{table}`	resets table captions to have default options

Table 14: Font Options

Option	Affected portion
`font=`	Affects entire caption (caption label, separator, and caption text).
`labelfont=`	Affects only caption label and separator
`textfont=`	Affects only caption text

Table 15: Possible Font Option Values

Action	`Option Value`	Description
Use all font defaults	`default`	Changes font family, shape, series, and size to defaults
Specify font family	`rm`	roman font family *(default)*
	`sf`	san sarif font family
	`tt`	`typewriter font family`
Specify font shape	`up`	upright font shape *(default)*
	`it`	*italic font shape*
	`sl`	*slanted font shape*
	`sc`	SMALL CAPS FONT SHAPE
Specify font series	`md`	medium font series *(default)*
	`bf`	**bold font series**
Specify font size	`scriptsize`	scriptsize font size
	`footnotesize`	footnotesize font size
	`small`	small font size
	`normalsize`	normal font size *(default)*
	`large`	large font size
	`Large`	Large font size

Table 16: captionsetup Vertical Space Options

Keyword	Value	Description
`aboveskip=`	`<amount>`	*(default is 10pt)* Sets the vertical spacing between the caption and the figure/table contents. Normally, this space is placed above the caption, but when `position=top` then the `aboveskip=` spacing is placed below the caption.
`belowskip=`	`<amount>`	*(default is 0pt)* Sets the vertical spacing in the direction away from the figure/table contents. Normally, this space is placed below the caption, but when `position=top` then `belowskip=` spacing is placed above the caption.
`position=`	`bottom`	*(default)* Places the `aboveskip=` spacing above the caption and the `belowskip=` spacing below the caption.
	`top`	This caption reverses the `aboveskip=` and `belowskip=` spacings (to accommodate captions at the top of floats).
`parskip=`	`<amount>`	*(default is 0pt)* The amount of vertical space inserted between a caption's paragraphs. (This option has no effect on captions with only one paragraph).

Table 17: captionsetup Label and Separator Options

Keyword	Value	Description
`labelformat=`	`default`	*(default)* Caption label is typeset as specified in the document class.
	`simple`	The caption label is typeset as a name and a number. For example "Figure 9".
	`parens`	The number is typeset inside parentheses. For example, "(9)".
	`empty`	The caption label is empty (no "Figure", no number). This is usually used with `labelsep=none` to also eliminate the caption separator.
`labelsep=`	`colon`	*(default)* The caption separator is a colon and a space
	`period`	The caption separator is a period and a space
	`space`	The caption separator is a single space
	`quad`	The caption separator is a `\quad`
	`newline`	The caption separator is a `\newline`
	`none`	No caption separator. Usually used only with `labelformat=empty`

Table 18: captionsetup Formatting Options

Keyword	Value	Description
`format=`	`plain`	*(default)* Captions are typeset as normal paragraphs.
	`hang`	The caption's 2nd and subsequent lines are indented. It generally causes the first character of the 2nd line to be horizontally aligned with the first character of the caption text on the first line.
`justification=`	`justified`	*(default)* Caption is typeset as a regular paragraph.
	`centerlast`	Last line of caption is horizontally centered.
	`centerfirst`	First line of caption is horizontally centered.
	`centering`	Each line of caption is horizontally centered.
	`Centering`	Same as `centering` except the TEX word-breaking algorithm is used.
	`raggedright`	Each line is left-justified, leaving a ragged right margin.
	`RaggedRight`	Same as `raggedright` except the TEX word-breaking algorithm is used.
	`raggedleft`	Each line is right-justified, leaving a ragged left margin.
	`RaggedLeft`	Same as `raggedleft` except the TEX word-breaking algorithm is used.
`indentation=`	`<amount>`	*(default is 0pt)* Amount of *additional* indentation for the caption's 2nd and subsequent lines.
`hangindent=`	`<amount>`	*(default is 0pt)* Amount of *additional* indentation for the 2nd and subsequent lines of *each* paragraph in a caption. Note `hangindent=` does not apply to the first line of each paragraph. (If a caption contains only one paragraph, then `hangindent=` and `indentation=` are equivalent.)
`margin=`	`<amount>`	*(default is 0pt)* Both left and right margins are brought in by specified amount.
`width=`	`<amount>`	Sets caption width (left and right margins are brought in equal amounts). If both `margin=` and `width=` values are specified, the last option specified applies.
`singlelinecheck=`	`true`	*(default)* If the caption fits on a single line, then the caption is centered regardless of `justification=` value.
	`false`	Format of `justification=` is applied to single line captions.

20.4 Caption Package Examples

20.4.1 Caption Package Font Options

Table 14 shows the three options the caption package provides for customizing a caption's fonts, where "label", "separator", "caption text" are defined in Figure 16 on page 76. These three options can change any combination of font family, shape, series, and size, as shown in Table 15.

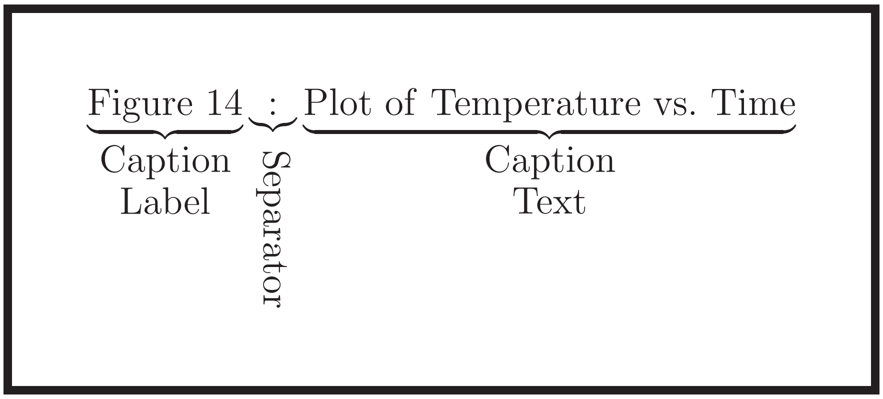


Figure 16: Definition of Caption Label, Separator, Text

Font Example #1

The command

```
\captionsetup{font={default,Large,bf}}
\caption{This is Caption Font Example \#1}
```

sets all caption font characteristics to their defaults, then gives entire figure caption (label, separator, and caption text) Large font size and bold font series, as shown in Figure 17. The font family and shape remain set to their defaults.

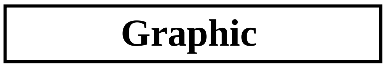


Figure 17: This is Caption Font Example #1

Note that the above example affects both figure and table captions. Adding a `[figures]` optional argument to the `\captionsetup` command

```
\captionsetup[figures]{font={default,Large,bf}}
```

causes only the figure captions to be modified.

Font Example #2

The command

```
\captionsetup{font=default, textfont={scriptsize,sf}}
\caption{This is Caption Font Example \#2}
```

sets all figure-caption font characteristics to their defaults, then gives the caption text scriptsize font size and san serif font family, as shown in Figure 18.

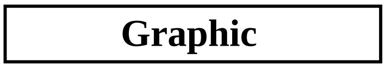


Figure 18: This is Caption Font Example #2

Font Example #3

The command

```
\captionsetup{font={default,Large}, labelfont=bf, textfont=sl}
\caption{This is Caption Font Example \#3}
```

sets all figure-caption font characteristics to their defaults, then gives the entire caption (label, separator, and caption text) Large font size, gives only the caption label and separator bold font series, and gives only the caption text slanted font shape, as shown in Figure 19.

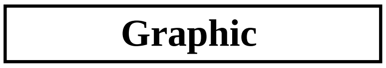


Figure 19: *This is Caption Font Example #3*

20.4.2 Caption Vertical Package Spacing Options

As their names suggest, the space above the caption is specified by the `aboveskip` option while the space below the caption is specified by the `belowskip` option. However, this is only the case when the default `position=bottom` option is used. The `position=top` option reverses the meanings of the `aboveskip` and `belowskip` options. It can get confusing if the user has customized `aboveskip` and `belowskip` for a top-caption and then wants a bottom caption, in which case the `position=top` option would have to be used to produce a bottom caption.

To avoid confusion, the following procedure should be used

1. Think of `aboveskip=` option as the caption/float spacing. That is, the space between the caption and the float contents should be specified in the `aboveskip=` option.
2. Think of `belowskip=` option as the caption/text spacing. That is, the space between the caption and the surrounding text should be specified in the `belowskip=` option.
3. Use `position=bottom` for captions at the bottom of floats. Use `position=top` for captions at the top of floats.

Caption Vertical Spacing Example #1

The `aboveskip=` option modifies the spacing above a caption (the default spacing is 10pt). The following code

```
\captionsetup{aboveskip=1cm}
\caption{Vertical Spacing Example \#1}
```

causes produces a 1cm spacing above the caption as shown in Figure 20.

Graphic

Figure 20: Vertical Spacing Example #1

Caption Vertical Spacing Example #2

The previous example showed how the `aboveskip=` option modifies the spacing above a caption. This example shows how the `position=top` reverses the meaning of `aboveskip` and `belowskip`.

The following code

```
\captionsetup{aboveskip=1cm,position=top}
\caption{Vertical Spacing Example \#2}
```

causes produces a 1cm spacing below the caption as shown in Table 19

Table 19: Vertical Spacing Example #2

a	b
c	d

Caption Vertical Spacing Example #3

The previous example showed the `position=top` reverses the meaning of `aboveskip` and `belowskip` for use in placing a caption above tables.

If table captions will be placed above the table while most figure captions will be placed below the figure, it is convenient to make the top position the default for table captions

```
\captionsetup{aboveskip=1cm,belowskip=0pt}
\captionsetup[figure]{position=bottom}
\captionsetup[table]{position=top}
...
\caption{Table for Vertical Spacing Example \#3}
\caption{Figure for Vertical Spacing Example \#3}
```

causes produces a 1cm spacing below table captions (as shown in Table 20) and a 1cm spacing above figure captions (as shown in Figure 21).

Table 20: Table for Vertical Spacing Example #3

a	b
c	d

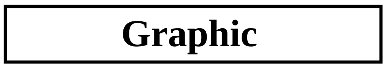


Figure 21: Figure for Vertical Spacing Example #3

20.4.3 Caption Package Label Options

Caption Label Example #1

The following code

```
\captionsetup{labelformat=simple}
\caption{This is Caption Label Example \#1}
```

define the label format to be `simple`, as shown in Figure 22. The `simple` format is usually the same as the `default`, although the document class could define `default` differently.

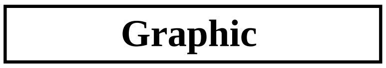


Figure 22: This is Caption Label Example #1

Caption Label Example #2

The following code

```
\captionsetup{labelformat=parens}
\caption{This is Caption Label Example \#2}
```

define the label format to be `parens`, as shown in Figure 23. This format has no Figure name, and instead just has the figure number surrounded by parentheses.

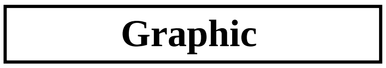


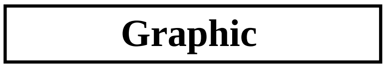
Figure (23): This is Caption Label Example #2

Caption Label Example #3

The following code

```
\captionsetup{labelformat=empty, labelsep=none}
\caption{This is Caption Label Example \#3}
```

define the label format to be `empty`, as shown in Figure 23. Since no figure number is displayed, it is difficult to determine that it is Figure 23.



This is Caption Label Example #3

Caption Label Example #4

The following code

```
\captionsetup{labelformat=default, labelsep=period}
\caption{This is Caption Label Example \#4}
```

changes the caption separator from the default colon to a period, as shown in Figure 25.

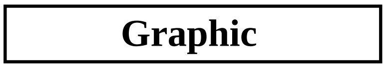


Figure 25. This is Caption Label Example #4

Caption Label Example #5

The following code

```
\captionsetup{labelformat=default, labelsep=newline}
\caption{This is Caption Label Example \#5}
```

changes the caption separator to a newline, as shown in Figure 25.

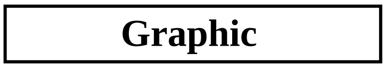


Figure 26
This is Caption Label Example #5

20.4.4 Caption Package Formatting Options

Formatting Example #1: Caption Width

The following code

```
\captionsetup{width=3in}
\caption{This is an example of customizing the caption width}
```

causes the caption to be typeset in a 3-inch wide column, as shown in Figure 27.

Graphic

Figure 27: This is an example of customizing the caption width

---

Formatting Example #2: Default Format

Figure 28 - Figure 36 on Page 82 show examples of the seven possible `justification=` values used with `format=default`. These figures are produced by the following code

```
\captionsetup{format=default,justification=justified}
\caption{Caption with default format and justified justification.
         Caption with default format and justified justification.
         Caption with default format and justified justification.}
...
\captionsetup{format=default,justification=centering}
\caption{Caption with default format and centering justification.
         Caption with default format and centering justification.
         Caption with default format and centering justification.}
...
\captionsetup{format=default,justification=centerlast}
\caption{Caption with default format and centerlast justification.
         Caption with default format and centerlast justification.
         Caption with default format and centerlast justification.}
...
\captionsetup{format=default,justification=centerfirst}
\caption{Caption with default format and centerfirst justification.
         Caption with default format and centerfirst justification.
         Caption with default format and centerfirst justification.}
...
\captionsetup{format=default,justification=raggedright}
\caption{Caption with default format and raggedright justification.
         Caption with default format and raggedright justification.
         Caption with default format and raggedright justification.}
...
\captionsetup{format=default,justification=RaggedRight}
\caption{Caption with default format and RaggedRight justification.
         Caption with default format and RaggedRight justification.
         Caption with default format and RaggedRight justification.}
...
\captionsetup{format=default,justification=raggedleft}
\caption{Caption with default format and raggedleft justification.
         Caption with default format and raggedleft justification.
         Caption with default format and raggedleft justification.}
```

As can be seen in Figure 28 - Figure 36, the above code causes the first line of the caption to be formatted the same as all of the other lines.

Formatting Example #3: Hang Format

The previous example used the `format=default` option. This previous example creates Figure 37 - Figure 45, showing the seven possible `justification=` values used with `format=hang`. These figures are produced by the following code

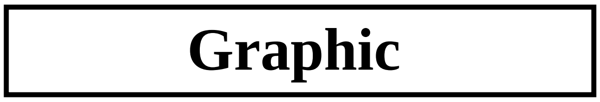


Figure 28: Caption with default format and justified justification. Caption with default format and justified justification. Caption with default format and justified justification.

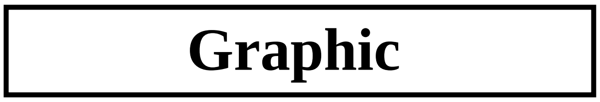


Figure 29: Caption with default format and centerlast justification. Caption with default format and centerlast justification. Caption with default format and centerlast justification.

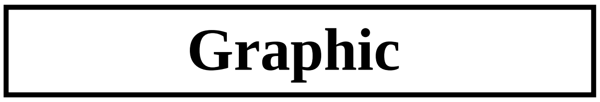


Figure 30: Caption with default format and centerfirst justification. Caption with default format and centerfirst justification. Caption with default format and centerfirst justification.

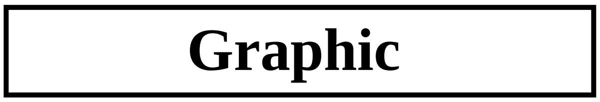


Figure 31: Caption with default format and centering justification. Caption with default format and centering justification. Caption with default format and centering justification.

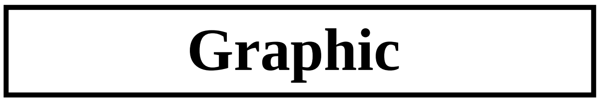


Figure 32: Caption with default format and Centering justification. Caption with default format and Centering justification. Caption with default format and Centering justification.

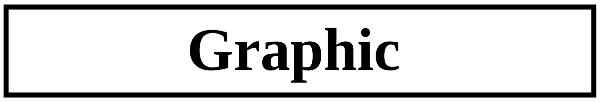


Figure 33: Caption with default format and raggedright justification. Caption with default format and raggedright justification. Caption with default format and raggedright justification.

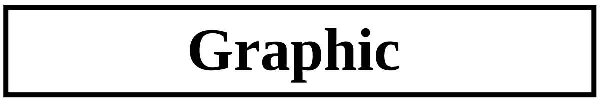


Figure 34: Caption with default format and RaggedRight justification. Caption with default format and RaggedRight justification. Caption with default format and RaggedRight justification.

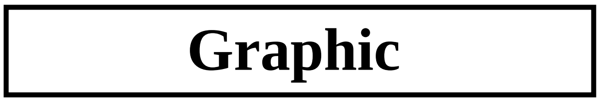


Figure 35: Caption with default format and raggedleft justification. Caption with default format and raggedleft justification. Caption with default format and raggedleft justification.

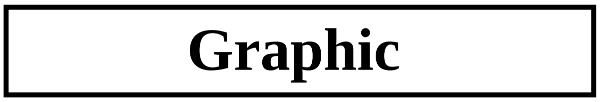


Figure 36: Caption with default format and RaggedLeft justification. Caption with default format and RaggedLeft justification. Caption with default format and RaggedLeft justification.

Graphic

Figure 37: Caption with hang format and justified justification. Caption with hang format and justified justification. Caption with hang format and justified justification.

Graphic

Figure 38: Caption with hang format and centerlast justification. Caption with hang format and centerlast justification. Caption with hang format and centerlast justification.

Graphic

Figure 39: Caption with hang format and centerfirst justification. Caption with hang format and centerfirst justification. Caption with hang format and centerfirst justification.

Graphic

Figure 40: Caption with hang format and centering justification. Caption with hang format and centering justification. Caption with hang format and centering justification.

Graphic

Figure 41: Caption with hang format and Centering justification. Caption with hang format and Centering justification. Caption with hang format and Centering justification.

Graphic

Figure 42: Caption with hang format and raggedright justification. Caption with hang format and raggedright justification. Caption with hang format and raggedright justification.

Graphic

Figure 43: Caption with hang format and RaggedRight justification. Caption with hang format and RaggedRight justification. Caption with hang format and RaggedRight justification.

Graphic

Figure 44: Caption with hang format and raggedleft justification. Caption with hang format and raggedleft justification. Caption with hang format and raggedleft justification.

Graphic

Figure 45: Caption with hang format and RaggedLeft justification. Caption with hang format and RaggedLeft justification. Caption with hang format and RaggedLeft justification.

```
\captionsetup{format=hang,indention=0pt,justification=justified}
\caption{Caption with hang format and justified justification.
         Caption with hang format and justified justification.
         Caption with hang format and justified justification.}
...
\captionsetup{format=hang,indention=0pt,justification=centering}
\caption{Caption with hang format and centering justification.
         Caption with hang format and centering justification.
         Caption with hang format and centering justification.}
...
\captionsetup{format=hang,indention=0pt,justification=centerlast}
\caption{Caption with hang format and centerlast justification.
         Caption with hang format and centerlast justification.
         Caption with hang format and centerlast justification.}
...
\captionsetup{format=hang,indention=0pt,justification=centerfirst}
\caption{Caption with hang format and centerfirst justification.
         Caption with hang format and centerfirst justification.
         Caption with hang format and centerfirst justification.}
...
\captionsetup{format=hang,indention=0pt,justification=raggedright}
\caption{Caption with hang format and raggedright justification.
         Caption with hang format and raggedright justification.
         Caption with hang format and raggedright justification.}
...
\captionsetup{format=hang,indention=0pt,justification=RaggedRight}
\caption{Caption with hang format and RaggedRight justification.
         Caption with hang format and RaggedRight justification.
         Caption with hang format and RaggedRight justification.}
...
\captionsetup{format=hang,indention=0pt,justification=raggedleft}
\caption{Caption with hang format and raggedleft justification.
         Caption with hang format and raggedleft justification.
         Caption with hang format and raggedleft justification.}
```

As can be seen in Figure 37 - Figure 45, the `format=hang` option in the above code causes the second and subsequent lines to have additional indentation.

20.5 Further Customization

The caption has additional feature for users who want additional customization. This section provides a brief description, with detailed instruction available in [12].

20.5.1 Caption Styles

Users can define a collection of caption options, called *caption styles*. The entire collection of options can be specified with a single option. For example, the caption package automatically defines a style named `default` such that

```
\captionsetup{style=default}
```

is equivalent to

```
\captionsetup{font=default,       labelfont=default,
              textfont=default,   parskip=0pt,
              labelformat=simple, labelsep=colon,
              format=default,     indentation=0pt,
              hangindent=0pt,     margin=0pt,
              parinident=0pt,     justification=justified,
              singlelinecheck=true}
```

The `\DeclareCaptionStyle` can be used in the document preamble to define other caption styles. These styles can either explicitly define all of the parameters or start with the `default` style and modify only the unique option values.

For example, the following `\DeclareCaptionStyle` command in the document's preamble

```
\DeclareCaptionStyle{BigLeft}{style=default, labelsep=period,
                              font=Large,    labelfont=bold,
                              justification=RaggedRight,
                              singlelinecheck=false}
```

allows the `BigLeft` style to be referenced by

```
\captionsetup{style=BigLeft}
\caption{This Caption uses BigLeft Style}
```

as shown in Figure 46.

Graphic

Figure 46
This Caption uses BigLeft Style

20.5.2 Additional Option Values

The caption package provides commands such as

```
\DeclareCaptionFont
\DeclareCaptionLabelSeparator
\DeclareCaptionLabelFormat
\DeclareCaptionFormat
\DeclareCaptionLabelJustification
```

to provide additional option values. These commands can only be issued in the document's preamble.

Option Definition Example #1

Table 15 on page 73 defines the possible font options that can be used by the `font=`, `labelfont=`, and `textfont=` options. The `\DeclareCaptionFont` allows the user to define additional values that can be used by these options. For example, the following command in the document's preamble

```
\DeclareCaptionFont{BigAndBold}{\Large\bfseries}
```

defines a `BigAndBold` font such that the following code

```
\captionsetup{font=BigAndBold}
\caption{This Caption uses a Custom Font}
```

produces the caption in Figure 47

Graphic

Figure 47: This Caption uses a Custom Font

Option Definition Example #2

Table 17 on page 74 describes how the `labelformat=` option controls how the "Figure 33" portion of the caption is displayed. The `\DeclareCaptionLabelFormat` allows the user to define additional `labelformat=` options. The symbols `#1` and `#2` are used in the definition to specify where the "Figure" and Figure Number are inserted. For example the following command in the document's preamble

```
\DeclareCaptionLabelFormat{hash}{#1 {\#}#2}
```

defines a `hash` formatting option such that a hash mark # is placed between just before the figure number. However, this definition has a flaw in that the space in the definition after `#1` is not desired should `#1` be empty. The `\bothIfFirst` command typesets both of its arguments if the first argument exists, otherwise neither argument is typeset. Similarly, the `\bothIfSecond` command typesets both of its arguments if the second argument exists, otherwise neither argument is typeset. The new definition using `\bothIfFirst` is

```
\DeclareCaptionLabelFormat{hash}{\bothIfFirst{#1}{ }{\#}#2}
```

This definition, when placed in the document's preamble allows the following code

```
\captionsetup{labelformat=hash}
\caption{This Caption has a Custom Label Format}
```

to produce the caption in Figure 48.

Graphic

Figure #48: This Caption has a Custom Label Format

Option Definition Example #3

Table 17 on page 74 defines the possible values for the `labelset=` option. The `\DeclareCaptionLabelSeparator` command allows users to define additional values for the `labelset=` option. For example, the following command in the document's preamble

```
\DeclareCaptionLabelSeparator{arrow}{\quad\ensuremath{\Rightarrow}\quad}
```

defines an `arrow` label separator such that the following code

```
\captionsetup{labelsep=arrow}
\caption{This Caption has a Custom Label Separator}
```

produce the caption in Figure 49.

Graphic

Figure 49 $\quad\Rightarrow\quad$ This Caption has a Custom Label Separator

---

Option Definition Example #4

Table 18 on page 75 specifies the values that can be used by the `format=` option. The `\DeclareCaptionFormat` command allows users to define additional values for the `format=` option.

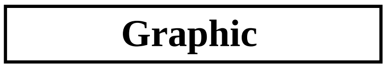
The symbols `#1`, `#2`, `#3` are used in the definition to specify where the various building blocks appear, where `#1` represent the caption label, `#2` represents the caption separator, and `#3` represents the caption text (where these terms are defined in Figure 16 on page 76). For example, the following command in the document's preamble

```
\DeclareCaptionFormat{reverse}{#3#2\ensuremath{\ll}#1\ensuremath{\gg}}
```

defines the `reverse` format with the caption text appearing first, followed by the separator, and then the caption label surrounded by double angle brackets `\ll` and `\gg`. The following code

```
\captionsetup{format=reverse,labelsep=empty}
\caption{This Caption has a Custom Format}
```

produces the caption in Figure 50.



This Caption has a Custom Format
≪Figure 50≫

---

21 Non-Floating Figures

Since non-floating figures can produce large sections of vertical whitespace, non-floating figures are generally considered poor typesetting style. Instead, users are strongly encouraged to use the figure environment's `[!ht]` optional argument which moves the figure only if there is not enough room for it on the current page.

As described in Section 17, LaTeX allows figures and tables to "float" to improve the document's formatting. Occasionally, it is desirable to have a figure appear *exactly* where it appears in the LaTeX source. Although the `\caption` command can only be used `figure` and `table` environments, the `caption` package defines the `\captionof` command which takes two arguments: the type of caption (table, figure, etc) and the caption text, allowing the `\captionof` command can be used outside of figure and table environments. Using

```
\captionof{figure}{caption text}
```

creates a figure caption, regardless of whether it appears in a figure environment. Likewise,

```
\captionof{table}{caption text}
```

creates a table caption, regardless of its location. The following commands

```
This is the text before the figure.
\\[\intextsep]
   \begin{minipage}{\linewidth}
       \centering
       \includegraphics[width=2in]{graphic}%
       \captionof{figure}{This is a non-floating figure}
       \label{fig:non:float}
   \end{minipage}
\\[\intextsep]
This is the text after the figure.
```

create a non-floating figure. Notes on non-floating figures:

- The `minipage` environment is needed to prevent a page break within the figure.
- The `\\[\intextsep]` commands start new lines and add vertical space before and after the figure. Any amount of space can be used, `\intextsep` (see Section 19.1 on page 64) was used to make the non-floating figure spacing consistent with floating figure spacing.
- Normally, figures are placed on the page in the same order they were submitted to the figure queue. However, non-floating figures are placed immediately, leapfrogging any unprocessed figures sitting in the figure queue. If this happens, the figures do not appear in numerical order[34]. To avoid these out-of-order figures, force all floating figures to be processed by issuing a `\clearpage` or `\FloatBarrier` command before the non-floating figure (see Section 17.3 on page 59).
- The `\captionof` command is also useful for creating marginal figures (Section 22 on page 89), and creating a table beside a figure (Section 30 on page 109).

21.1 Non-floating Figures without the caption package

As described above, the caption package's `\captionof` command creates captions outside of figure/table environments. This section describes how to do this without using the caption package.

The `\caption` command can be used in `figure` and `table` environments because these environments define the internal command `\@captype` to "figure" and "table", respectively. By defining `\@captype`, the `\caption` command can be used outside of figure and table environments. A `\makeatletter`–`\makeatother` pair *must* enclose `\@captype` to allow `@` to be used in a command name. While this can be done manually each time by

```
\includegraphics{file}
\makeatletter\def\@captype{figure}\makeatother
\caption{This is the caption}
```

it is easier to define a command to do this. Including the following commands in the document's preamble

```
\makeatletter
   \newcommand\figcaption{\def\@captype{figure}\caption}
   \newcommand\tabcaption{\def\@captype{table}\caption}
\makeatother
```

defines the `\figcaption` and `\tabcaption` commands. Using `\figcaption` creates figure captions, regardless of whether it appears in a figure environment. Likewise, `\tabcaption` creates table caption, regardless of it location.

21.2 The float Package's [H] Placement Option

The float package[35] adds an `[H]` option to the `figure` environment which produces a non-floating figure. The following code

[34] In these situations, the Table of Figures lists the figures in order of appearance, not in numerical order.

[35] The float package allows users to define new types of floats, such as "Program" or "Algorithm." It also defines optional boxed and ruled float styles. These optional float styles redefine the `\caption` command such that the caption is always typeset at a particular location, regardless of where the `\caption` command is located, preventing construction of side-by-side and other complex figures.

```
\usepackage{float}
...
\begin{figure}[H]
     .....
\end{figure}
```

produces a non-floating figure.

When the [H] figure does not fit on a page, the figure is moved to the top of the next page. If there were any footnotes on the first page, they appear immediately after the text instead of at the bottom of the page. If this behavior is undesirable, then the \captionof command described in Section 21 on page 87 can be used instead of the float package's [H] placement option.

22 Marginal Figures

The \marginpar command places notes in the margin of the document. The marginal notes are placed in the right margin (or the outside margin for twoside documents) unless the \reversemarginpar command is used (as it is in this document). The width of the marginal column is controlled by the \marginparwidth length, while the horizontal spacing between the text and the marginal notes is controlled by the \marginparsep length.

Marginal notes are placed such that the baseline of their first line is vertically aligned with the baseline of the text which contains the \marginpar command.

Marginal notes are never broken across a page; if a marginal note starts near the bottom of the page, it continues into the bottom margin. If the previous marginal note would interfere with a marginal note, LaTeX "bumps" the latter marginal note downward. Marginal notes cannot be bumped to the next page; they are instead bumped into the bottom margin. As a result, the position of the marginal notes may have to be adjusted before the final printing to avoid marginal notes near page breaks.

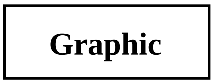


Figure 51: This is a Marginal Figure

Since the figure environment cannot be used in a marginal note, floating marginal figures are not possible. However, the \captionof command defined in Section 21 on page 87 can be used to construct a non-floating marginal figure. For example, Figure 51 was produced by

```
...to construct a non-floating marginal figure.
\marginpar{\centering
           \includegraphics[width=\marginparwidth]{graphic}%
           \captionof{figure}{This is a Marginal Figure}
           \label{fig:marginal:fig} }
For example, \Figref{fig:marginal:fig} was...
```

The bottom of the graphic in Figure 51 is aligned with the text baseline where the \marginpar command is located. Notes on marginal figures:

- Since captions for marginal figures generally are narrow, using either of the caption package commands

  ```
  \captionstyle{justification=raggedright}
  \captionstyle{justification=raggedleft}
  ```

  before the \caption command may provide better caption formatting. Additionally, the caption package command

  ```
  \captionsetup{font=small}
  ```

  can be used to decrease the size of the caption font. See Section 20 on page 69 for caption information.

- Like the non-floating figures in Section 21 on page 87, the marginal figures are placed ahead of any unprocessed floats. Thus, a `\clearpage` or `\FloatBarrier` command must be issued before the marginal note if one wants to keep the figures in order.
- Marginal notes are placed by the routine which also places figures and tables. If many figures, tables, and marginal notes are being used, it is possible to exceed the number of unprocessed floats permitted by LaTeX. The morefloats package can mitigate these problems (see Section 17.4 on page 60).

23 Wide Figures

Typesetting readability rules limit the number of characters in a line of text. Unless a large font or two columns are used, these readability rules result in wide margins (especially when using 8.5 x 11 inch letter paper). Section 22 demonstrated how these wide margins can be used for marginal figures. Another option is to construct a regular floating figure which extends into one or both margins. This is done by placing a wide list environment inside the figure. For example, a `narrow` environment can be defined by including the following code in the preamble of your document

```
\newenvironment{narrow}[2]{%
  \begin{list}{}{%
  \setlength{\topsep}{0pt}%
  \setlength{\leftmargin}{#1}%
  \setlength{\rightmargin}{#2}%
  \setlength{\listparindent}{\parindent}%
  \setlength{\itemindent}{\parindent}%
  \setlength{\parsep}{\parskip}}%
\item[]}{\end{list}}
```

For example, any text which occurs between `\begin{narrow}{1in}{2in}` is indented by 1 inch on the left side and 2 inches on the right side. When negative lengths are used, the contents extend beyond the margins.

23.1 Wide Figures in One-sided Documents

The following code uses this `narrow` environment to make the figure extend 1 inch into the left margin, producing Figure 52.

```
\begin{figure}
\begin{narrow}{-1in}{0in}
   \includegraphics[width=\linewidth]{wide}
   \caption{This is a wide figure}
\end{narrow}
\end{figure}
```

Figure 52: This is a wide figure

When marginal notes are used, it may be desired to make the wide figure extend exactly to the edge of the marginal notes (making the figure width be `\linewidth`

+ \marginparwidth + \marginparsep). This can be done by defining a new length \marginwidth and setting it to be \marginparwidth + \marginparsep. For example,

```
\newlength{\marginwidth}
\setlength{\marginwidth}{\marginparwidth}
\addtolength{\marginwidth}{\marginparsep}
```

then use {-\marginwidth} in the \begin{narrow} argument.

23.2 Wide Figures in Two-sided Documents

For two-sided documents, it may be desired to extend the wide figures into the binding-side margin (i.e., the left margin for odd pages and the right margin for even pages). In these cases, the ifthen package's \ifthenelse command can be used to choose between odd-page code and even-page code. For example,

```
\usepackage{ifthen}
...
\begin{figure}
  \ifthenelse{\isodd{\pageref{fig:wide}}}%
  {% BEGIN ODD-PAGE FIGURE
   \begin{narrow}{0in}{-1in}
      \includegraphics[width=\linewidth]{file}
      \caption{Figure Caption}
      \label{fig:wide}
    \end{narrow}
  }% END ODD-PAGE FIGURE
  {% BEGIN EVEN-PAGE FIGURE
   \begin{narrow}{-1in}{0in}
      \includegraphics[width=\linewidth]{file}
      \caption{Figure Caption}
      \label{fig:wide}
    \end{narrow}
  }% END EVEN-PAGE FIGURE
\end{figure}
```

Since the \pageref command is used as input to \ifthenelse, the figure may not be properly typeset until LaTeX is run enough times to cause the cross-references to converge.

24 Landscape Figures

In a document with portrait orientation, there are three methods for producing figures with landscape orientation.

1. The lscape package provides a landscape environment, which treats the left edge of the paper as the top of the page, causing any text, tables, or figures in the landscape environment to have landscape orientation.
2. The rotating package provides a sidewaysfigure environment which is similar to the figure environment except that the figures have landscape orientation.
3. The rotating package provides a \rotcaption command which is similar to the \caption command except that caption has landscape orientation.

Differences between methods

- Both options 1 and 2 place the landscape figure on a separate page. Option 3 produces an individual float which need not be on its own page.

- While Option 2 produces only rotated figures, the `landscape` environment in Option 1 is a general-purpose environment, which can produce landscape pages containing any combination of text, tables, and figures. The `landscape` environment can page-breaking capability, so multiple landscape pages can be produced[36].
- The full-page figure produced by Option 2 floats to provide better document formatting, while the figure produced by Option 1 cannot float[37].
- Since Options 1 and 3 use the `figure` environment, they can be used in conjunction with the endfloat package (see Section 19.6 on page 68).
- Options 1 and 2 are best suited for side-by-side landscape graphics (for side-by-side methods see Section 28 on page 104).

24.1 Landscape Environment

The lscape package (which is part of the standard "graphics bundle" distributed with LaTeX) defines the `landscape` environment, which provides a method of placing landscape pages in a portrait document. The landscape pages are rotated such that the left edge of the portrait page is the top edge of the landscape page.

Entering `\begin{landscape}` prints all unprocessed portrait floats and then switches to landscape orientation. Likewise, `\end{landscape}` prints all unprocessed landscape floats and then switches back to portrait orientation.

The entire contents of the `landscape` environment is typeset with landscape orientation. This may include any mixture of text, figures, and tables. If the `landscape` environment contains only a figure environment

```
\begin{landscape}
  \begin{figure}
     \centering
     \includegraphics[width=4in]{graphic}
     \caption{Landscape Figure}
  \end{figure}
\end{landscape}
```

the `landscape` environment produces a landscape figure. Note that since the `landscape` environment starts a new page, it may result in a partially-blank page.

24.2 Sidewaysfigure Environment

The rotating package provides the `sidewaysfigure` environment which produces figures with landscape orientation[38]. For example

```
\begin{sidewaysfigure}
   \centering
   \includegraphics[width=4in]{graphic}
   \caption{Sidewaysfigure Figure}
\end{sidewaysfigure}
```

produces Figure 53.

---

[36] The `landscape` environment works very well with the longtable package to produce multiple-page landscape tables.

[37] Figures issued in the `landscape` environment can float within the landscape pages

[38] The rotating package also provides a `sidewaystable` environment for producing tables with landscape orientation.

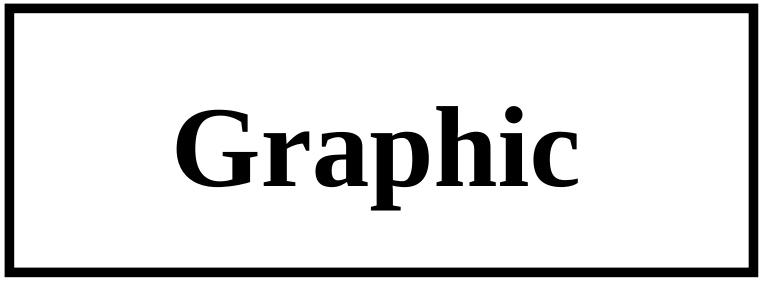


Figure 53: Sidewaysfigure Figure

Unlike the landscape environment, the figure produced by sidewaysfigure can float within the portrait pages to avoid the partially-blank page that the landscape environment may produce. Note that the landscape environment is much more flexible, allowing the landscape pages to consist of a mixture of text, tables, and figures.

The default orientation of the figures produced by sidewaysfigure depends on whether the document is processed with the oneside or twoside documentclass option

- When the oneside option is chosen, the bottom of graphic is towards the right edge of the portrait page.
- When the twoside option is chosen, the bottom of graphic is towards the outside edge of the portrait page.

This default behavior can be overridden by options to the \usepackage{rotating} command.

```
\usepackage[figuresleft]{rotating}
```

causes the bottom of the sidewaysfigure graphics to be towards the left edge of the portrait page (regardless of oneside or twoside options). Similarly,

```
\usepackage[figuresright]{rotating}
```

causes the bottom of the sidewaysfigure graphics to be towards the right edge of the portrait page.

24.3 Rotcaption Command

The methods in Sections 24.1 and 24.2 both produce full-page landscape figures, which may not be necessary for smaller landscape figures. The rotating package's \rotcaption command can be used to construct smaller landscape figures. For example

```
\begin{figure}
  \centering
  \begin{minipage}[c]{1in}
     \hfill\includegraphics[width=2in,angle=90]{graphic}
  \end{minipage}%
  \hspace{0.2in}%
  \begin{minipage}[c]{0.5in}
     \captionsetup{width=2in}
     \rotcaption{This is a caption created by the Rotcaption command}
     \label{fig:rotcaption}
  \end{minipage}
\end{figure}
```

produces Figure 54.

The caption produced by \rotcaption is always rotated such that its bottom is towards the right edge of the paper. Unlike the methods in Sections 24.1 and 24.2, the \rotcaption command does not rotate the graphics. Therefore, the \includegraphics command in the above example requires the angle=90 option.

25 Captions Beside Figures

Although the caption of a figure is generally placed above or below the graphic, this section describes how to place the caption beside the graphic[39].

[39] Since the figure environment defined by the float package places the caption *below* the body, captions beside figures cannot be produced with the float package's figure environment. Other

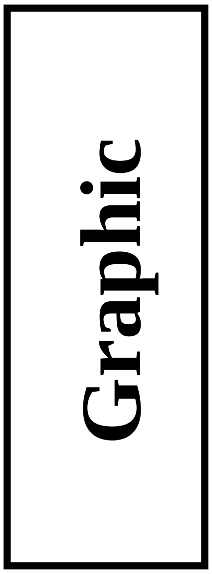


Figure 54: This is a caption created by the Rotcaption command

25.1 The Sidecap Package

The easiest way of constructing side captions is to use the sidecap package. When a `\caption` command is used in the SCfigure and SCtable environments defined by the sidecap package, the captions are automatically placed to the side of the contents of the environment. For example,

```
\usepackage{sidecap}
...
\begin{SCfigure}
   \includegraphics[width=3in]{graphic}
   \caption{This is a SCfigure}
\end{SCfigure}
```

produces Figure 55.

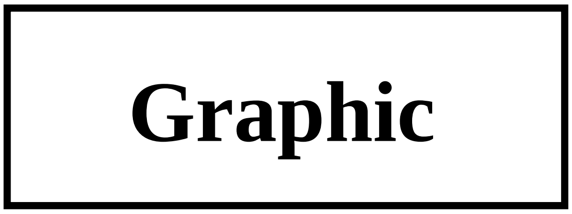


Figure 55: This is a SCfigure

The following four options can be specified in the `\usepackage` command

outercaption This option places the caption to the left for left (even) pages and on the right for right (odd) pages. (This is the default)

innercaption This option places the caption to the right for left (even) pages and on the left for right (odd) pages.

leftcaption This option places the caption on the left.

rightcaption This option places the caption on the right.

The `SCfigure` environment includes two optional arguments

aspects of the float package can be used as long as the `\restylefloat` command is not issued.

- The first optional argument specifies the relative width of caption compared to the figure. A large value (e.g., 100) reserves the maximum possible width. The default is 1.
- The second optional argument specifies the float position parameter (e.g. `[htp]` or `[!ht]` (see Section 17.2 on page 58).

25.2 Side Captions without Sidecap

If the sidecap package does not provide sufficient flexibility, users can produce side captions with the methods in this section. Section 25.2.1 shows how to place the caption to the left of the graphic. Placing the caption to the right of the graphic proceeds similarly. For `twoside` documents, Section 25.2.2 shows how to place the caption to the inside of the graphic (to the left of the graphic for odd pages and to the right of the graphic for even pages).

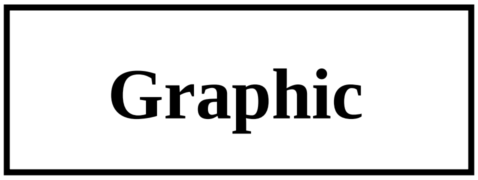
25.2.1 Caption to Left of Figure

The `\caption` command places the caption under the figure or table. Minipage environments can be used to trick the caption command into placing the caption beside the figure. For example, the commands

```
\begin{figure}
   \centering
   \begin{minipage}[c]{.45\linewidth}
      \centering
      \caption{Caption on the Side}
      \label{fig:side:caption}
  \end{minipage}%
  \begin{minipage}[c]{.45\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
  \end{minipage}
\end{figure}
```

produces Figure 56. It may be desirable to place a horizontal spacing command such as `\hfill` or `\hspace{.05\linewidth}` between the minipages.

Figure 56: Caption on the Side



The caption and graphic in Figure 56 are centered vertically. If it is instead desired to align the bottoms or tops of graphics and caption, see Section 11.4 on page 36.

25.2.2 Caption on Binding Side of Graphic

The above code for Figure 56 places the caption to the left of the graphic. For two-sided documents, it may be desired to place the caption on the binding side of the graphics. In these cases, the ifthen package's `\ifthenelse` command can be used to choose between odd-page code and even-page code. For example,

```
\usepackage{ifthen}
...
\begin{figure}
   \centering
```

```
\ifthenelse{\isodd{\pageref{fig:side:caption}}}
{% BEGIN ODD-PAGE FIGURE
   \begin{minipage}[c]{.45\linewidth}
      \centering
      \caption{Caption on the Side}
      \label{fig:side:caption}
   \end{minipage}%
   \hspace{0.05\linewidth}%
   \begin{minipage}[c]{.45\linewidth}
      \includegraphics[width=\linewidth]{graphic}
   \end{minipage}%
}% END ODD-PAGE FIGURE
{% BEGIN EVEN-PAGE FIGURE
   \begin{minipage}[c]{.45\linewidth}
      \includegraphics[width=\linewidth]{graphic}
   \end{minipage}%
   \hspace{0.05\linewidth}%
   \begin{minipage}[c]{.45\linewidth}
      \centering
      \caption{Caption on the Side}
      \label{fig:side:caption}
   \end{minipage}%
}% END EVEN-PAGE FIGURE
\end{figure}
```

produces a figure where the caption always appear on the binding side of the graphic.

26 Figures on Even or Odd Pages

The figure environment float-placement algorithm does not control whether a figure appears on an even or odd page. This section describes how to use the `\afterpage` command (part of the afterpage package) and the `\ifthenelse` command (part of the ifthen package) to place a figure onto an odd or even page.

The conventional method for creating figures is to put the graphics in a figure environment. However, since figure environments can float, there is no guarantee that a figure desired for an even-page won't end up on an odd page (or vice versa).

Instead, the `\captionof` command described in Section 21 can be used to create a figure without using a figure environment. The `\ifthenelse` command is then used to place the first graphic on the next even page. This requires repeating the graphics commands twice, once for the case of the next page being odd and once for the case of the next page being even. To simplify the resulting code, a `\leftfig` command is defined

```
\newcommand\leftfig{%
    \vspace*{\fill}%
    \centering
    \includegraphics{graphic}
    \captionof{figure}{This is on the left (even) page.}
    \vspace*{\fill}\newpage}
```

The left-page figures are then created using this newly-defined `\leftfig` command along with the `\afterpage` and `\ifthenelse` commands

```
\afterpage{\clearpage%
     \ifthenelse{\isodd{\value{page}}}%
         {\afterpage{\leftfig}}%
         {\leftfig}}
```

Notes about odd/even page placement:

- To force the figure to a right-hand (odd) page, reverse the order of the `\ifthenelse` arguments.

```
\afterpage{\clearpage%
      \ifthenelse{\isodd{\value{page}}}%
          {\leftfig}}%
          {\afterpage{\leftfig}}
```

- Because these are non-floating figures, the `\value{page}` command can be used to determine the current page. (This is not useful for floating figures since `\value{page}` is the current page when the figure environment is processed, not where it is placed.) Thus using `\value{page}` is better than `\pageref` since `\pageref` is only correct once the LaTeX references have converged).
- When using large figures, it is possible for a pagebreak to occur within the figure (e.g., between the graphic and the caption). The figure can be forced to stay together by enclosing it in a minipage environment

```
\newcommand\leftfig{%
        \vspace*{\fill}%
        \begin{minipage}{\linewidth}
           \centering
           \includegraphics{graphic}
           \captionof{figure}{This is on the left (even) page.}
        \end{minipage}
        \vspace*{\fill}\newpage}
```

- The `\afterpage` command can sometimes be flaky, in rare cases causing a "lost float" error. Removing the `\clearpage` before the `\ifthenelse` may help this situation.

```
\afterpage{\ifthenelse{\isodd{\value{page}}}%
          {\afterpage{\leftfig}}%
          {\leftfig}}
```

- In the above example, the figure uses the entire even page. To place the figure at the top of the even page, modify or remove the `\vspace*{\fill}` and `\newpage` commands

```
\newcommand\leftfig{%
    \centering
    \includegraphics{graphic}
    \captionof{figure}{This is at the top of the left (even) page.}
    \vspace{\floatsep}}
```

26.1 Figures on Facing Pages

To ease the comparison of two figures in a `twoside` document, it may be desirable to position the figures on facing pages. To do this, a procedure similar to the previous section's even/odd page-placement must be used. To simplify the resulting code, a `\facingfigures` command is defined as

```
\newcommand\facingfigures{%
     \vspace*{\fill}%
        \centering
        \includegraphics{left}
        \captionof{figure}{This is on the left (even) page.}
     \vspace*{\fill}\newpage\vspace*{\fill}%
        \centering
        \includegraphics{right}
        \captionof{figure}{This is on the right (odd) page.}
     \vspace*{\fill}\newpage}
```

The facing figures are then created using this `\facingfigures` command along with the `\afterpage` and `\ifthenelse` commands

```
\afterpage{\clearpage%
   \ifthenelse{\isodd{\value{page}}}%
      {\afterpage{\facingfigures}}%
      {\facingfigures}}
```

27 Boxed Figures

The term *Boxed Figure* usually refers to one of two situations

- A box surrounds the figure's graphic but not the figure's caption.
- A box surrounds the figure's graphic and its caption.

The basic method for boxing an item is to simply place the item inside an `\fbox` command, which surrounds the object with a rectangular box. The `fancybox` package provides boxes of different styles.

27.1 Box Around Graphic

Placing an `\fbox` command around the `\includegraphics` command produces a box around the included graphic. For example, the commands

```
\begin{figure}
   \centering
   \fbox{\includegraphics[totalheight=2in]{file}}
   \caption{Box Around Graphic, But Not Around Caption}
   \label{fig:boxed_graphic}
\end{figure}
```

place a box around the included figure, as shown in Figure 57.

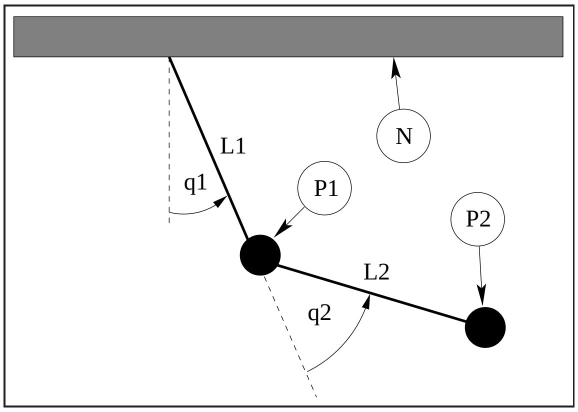


Figure 57: Box Around Graphic, But Not Around Caption

27.2 Box Around Figure and Caption

To include both the figure's graphic and its caption, one may be tempted to move the `\caption` command inside the `\fbox` command. However, this does not work because `\caption` can only be used in paragraph mode, while the contents of an `\fbox` command are processed in LR mode[40].

[40] LaTeX uses three modes: LR mode, paragraph mode, and math mode. See [1, pages 36,103-5].

Since the contents of minipage environments and `\parbox` commands are processed in paragraph mode, the `\caption` command can be included in the `\fbox` by enclosing the `\fbox` contents inside a minipage environment or a `\parbox` command. Since both minipages and parboxes require a width specification, there is no direct way to make the `\fbox` exactly as wide the graphic and caption.

For example, the commands

```
\begin{figure}
   \centering
   \fbox{ \begin{minipage}{4 in}
          \centering
          \includegraphics[totalheight=2in]{pend}
          \caption{Box Around Figure Graphic and Caption}
          \label{fig:boxed_figure}
   \end{minipage} }
\end{figure}
```

place a box around the figure's graphic and caption, as shown in Figure 58

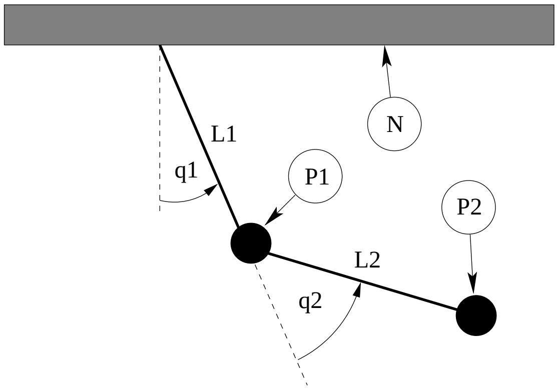


Figure 58: Box Around Figure Graphic and Caption

It is usually a trial-and-error process to determine a minipage width which causes the box to have a snug fit around the caption and graphic. This trial-and-error can be avoided by the following approaches.

1. Choose an arbitrary minipage width and force the graphic to be as wide as the minipage

   ```
   \includegraphics[width=\linewidth]{pend}
   ```

2. When it is desired to specify the graphic height, the proper minipage width can be calculated by placing the graphic in a box and measuring the height of the box.

   ```
   \newsavebox{\mybox}
   \newlength{\mylength}
   \sbox{\mybox}{\includegraphics[height=3in]{file}}
   \settowidth{\mylength}{\usebox{\mybox}}
   \begin{figure}
      \centering
      \fbox{ \begin{minipage}{\mylength}
           \centering
           \usebox{\mybox}
           \caption{Box Around Figure Graphic and Caption}
           \label{fig:boxed_figure}
   ```

```
      \end{minipage} }
   \end{figure}
```

3. To ensure a one-line caption, the minipage can be made as wide as the caption by estimating the caption width with a `\settowidth` command

```
\newlength{\mylength}
\settowidth{\mylength}{Figure XX:  Box Around Figure Graphic and Caption}
\fbox{ \begin{minipage}{\mylength}
...
```

27.3 Customizing fbox Parameters

In Figures 57 and 58, the box is constructed of 0.4 pt thick lines with a 3 pt space between the box and the graphic. These two dimensions can be customized by setting the LaTeX length variables `\fboxrule` and `\fboxsep`, respectively, with the `\setlength` command. For example, the commands

```
\begin{figure}
   \centering
   \setlength{\fboxrule}{3pt}
   \setlength{\fboxsep}{1cm}
   \fbox{\includegraphics[totalheight=2in]{pend}}
   \caption{Graphic with Customized Box}
   \label{fig:boxed_custom}
\end{figure}
```

place a box with 3 pt thick lines which is separated from the graphic by 1 centimeter, as shown in Figure 59

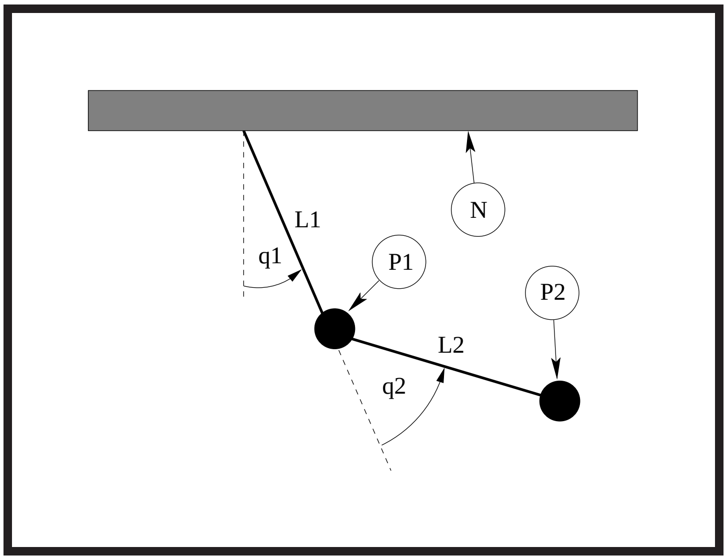


Figure 59: Graphic with Customized Box

27.4 The Fancybox Package

In Figures 57, 58, and 59, the `\fbox` command was used to place standard rectangular boxes around the figures. The `fancybox` package provides four commands `\shadowbox`, `\doublebox`, `\ovalbox`, and `\Ovalbox` which produce other types of boxes as shown in Table 21.

Like `\fbox`, the separation between these boxes and their contents is controlled by the LaTeX length `\fboxsep`. The length `\shadowsize` is set with the `\setlength`

Table 21: FancyBox Commands

Command	Parameters
`\shadowbox{Example}` Example	• The frame thickness is `\fboxrule`. • The shadow thickness is `\shadowsize` (which defaults to 4 pt).
`\doublebox{Example}` Example	• The inner frame thickness is .75`\fboxrule` • The outer frame thickness is 1.5`\fboxrule` • The spacing between the frames is 1.5`\fboxrule` + 0.5pt.
`\ovalbox{Example}` Example	• The frame thickness is `\thinlines` • Entering `\cornersize{x}` the diameter of the corners `x` times the minimum of the width and the height. The default is 0.5. • The `\cornersize*` command directly sets the corner diameter. For example, `\cornersize*{1cm}` makes the corner diameters 1 cm.
`\Ovalbox{Example}` Example	`\Ovalbox` is the same as `\ovalbox` except that the line thickness is controlled by `\thicklines`.

command, as was done for `\fboxrule` and `\fboxsep` in Section 27.3 on page 101. The lines for `\ovalbox` and `\Ovalbox` have thicknesses corresponding to the picture environment's `\thicklines` and `\thinlines`, which are *not* lengths and thus cannot be changed with the `\setlength` command. The values of `\thicklines` and `\thinlines` depend on the size and style of the current font. Typical values are 0.8 pt for `\thicklines` and 0.4 pt for `\thinlines`. For example, the commands

```
\begin{figure}
    \centering
    \shadowbox{ \begin{minipage}{3.5 in}
        \centering
        \includegraphics[totalheight=2in]{pend}
        \caption{Shadowbox Around Entire Figure}
        \label{fig:boxed_fancy}
    \end{minipage} }
\end{figure}
```

place a shadow box around the figure's graphic and caption, as shown in Figure 60.

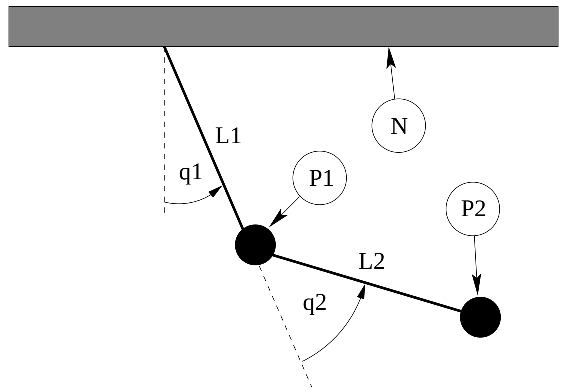


Figure 60: Shadowbox Around Entire Figure

Part V
Complex Figures

28 Side-by-Side Graphics

The commands necessary for side-by-side graphics depend on how the user wants the graphics organized. This section covers three common groupings of side-by-side graphics

1. The side-by-side graphics are combined into a single figure.
2. The side-by-side graphics each form their own figure (e.g., Figure 63 and Figure 64).
3. The side-by-side graphics each form a subfigure (e.g., Subfigure 65a and Subfigure 65b) which are part of a single figure (Figure 65).

This section describes the following two methods for constructing the three types of groupings

a) Successive `\includegraphics` commands.

b) Side-by-side minipages, each of which contains an `\includegraphics` command.

It is very important to understand the material in Section 2 on page 10 when constructing side-by-side figures. Side-by-side figures are created by placing boxes (either `\includegraphics` or minipages) beside each other on a line.

28.1 Side-by-Side Graphics in a Single Figure

The easiest method for creating side-by-side graphics in a single figure is successive `\includegraphics` commands, although using side-by-side minipages makes it easier to vertically align the graphics.

28.1.1 Using Side-by-Side includegraphics Commands

The following code

```
\begin{figure}
   \centering
   \includegraphics[width=1in]{graphic}%
   \hspace{1in}%
   \includegraphics[width=2in]{graphic}
   \caption{Two Graphics in One Figure}
\end{figure}
```

produces Figure 61 which is 4 inches wide (1 inch for the first graphic, 1 inch for the `\hspace`, and 2 inches for the second graphic) which is centered on the page. The `\hspace` command can be omitted or replaced with `\hfill`, which pushes the graphics to the margins (see Section 10.2 on page 32).

28.1.2 Using Side-by-Side Minipages

Placing the `\includegraphics` commands inside `minipage` environments provides the user more control over the graphics' vertical placement. For example

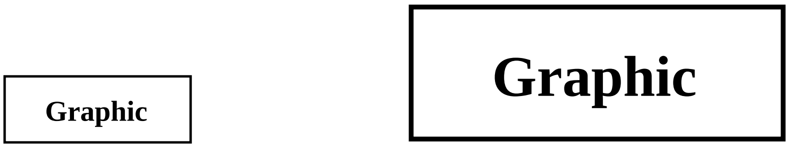


Figure 61: Two Graphics in One Figure

```
\begin{figure}
   \centering
   \begin{minipage}[c]{0.5\linewidth}
      \centering \includegraphics[width=1in]{graphic}
   \end{minipage}%
   \begin{minipage}[c]{0.5\linewidth}
      \centering \includegraphics[width=2in]{graphic}
   \end{minipage}
   \caption{Centers Aligned Vertically}
\end{figure}
```

produces Figure 62, which has vertically-centered graphics.

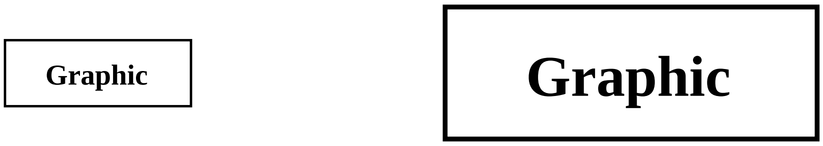


Figure 62: Centers Aligned Vertically

Notes on this example:

- Like any other LaTeX object, minipages are positioned such that their reference point is aligned with the current baseline. By default, minipages use the `[c]` option which places the reference point at the vertical center of the minipage, the `[t]` option places the reference point at the baseline of the minipage's top line and the `[b]` option places the reference point at the baseline of the minipage's bottom line (see Section 11.4 on page 36).
- The `%` after the first `\end{minipage}` command prevents an interword space from being inserted between the `minipage` boxes (see Section 10.2 on page 32).
- When the widths of the minipages do not add to 1.0`\linewidth`, the `\hspace` or `\hfill` commands can be used to specify horizontal spacing (see Section 10.2 on page 32).

28.2 Side-by-Side Figures

In the previous section, multiple `minipage` environments were used inside a `figure` environment to produce a single figure consisting of multiple graphics. Placing `\caption` statements inside the minipages makes the minipages themselves become figures. For example

```
\begin{figure}
   \centering
   %%----start of first figure----
   \begin{minipage}[t]{0.4\linewidth}
      \centering
      \includegraphics[width=1in]{graphic}
```

```
      \caption{Small Box} \label{fig:side:a}
   \end{minipage}%
   \hspace{1cm}%
   %%----start of second figure----
   \begin{minipage}[t]{0.4\linewidth}
      \centering
      \includegraphics[width=1.5in]{graphic}
      \caption{Big Box}  \label{fig:side:b}
   \end{minipage}
\end{figure}
```

produces Figures 63 and 64.

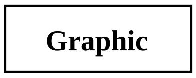


Figure 63: Small Box

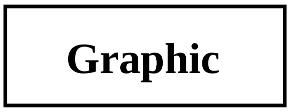


Figure 64: Big Box

Notes about this example:

- Although the above commands include *one* figure environment, the commands produce *two* figures because two `\caption` commands are used.
- The figures are put inside two minipages whose widths are 40% of the width of the figure environment that are separated by 1cm of horizontal space. (Note that comment characters after `\end{minipage}` and `\hspace{1cm}` ensure that the spacing is exactly 1cm by preventing interword spaces between the minipages and the horizontal space.)

  By default, the figure captions are typeset to the entire width of the minipage. The 1cm of horizontal space was used to ensure horizontal spacing between the captions (for longer captions and/or wider graphics).

  Alternatively, the caption widths could be limited by the caption package's `margin` or `width` keywords (see Table 18 on page 75).
- The `\centering` command immediately after `\begin{figure}` causes the group of two minipages and spacing to be centered in the figure environment.
- The `\centering` command inside the minipage causes the graphics to be centered within the minipage.

28.3 Side-by-Side Subfigures

It may be desirable to refer to side-by-side graphics both individually and as a group. The `\subfloat` command (from the subfig package, described in Section 32 on page 113) allows a group of graphics to be individually defined as subfigures that are defined to be part of a single figure. For example

```
\usepackage{subfig}
...
\begin{figure}
   \centering
   %%----start of first subfigure----
   \subfloat[Small Box with a Long Caption]{
        \label{fig:subfig:a}           %% label for first subfigure
        \includegraphics[width=1.0in]{graphic}}
   \hspace{1in}
   %%----start of second subfigure----
   \subfloat[Big Box]{
```

```
        \label{fig:subfig:b}          %% label for second subfigure
        \includegraphics[width=1.5in]{graphic}}
    \caption{Two Subfigures}
    \label{fig:subfig}                %% label for entire figure
\end{figure}
```

produces Figure 65. The commands used to individually and collectively reference the parts of Figure 65 are shown in Table 22.

(a) Small Box with a Long Caption

(b) Big Box

Figure 65: Two Subfigures

Table 22: Subfigure Reference Commands and Their Output for Figure 65 Example

Reference Command	Output
`\subref{fig:subfig:a}`	(a)
`\subref*{fig:subfig:a}`	a
`\ref{fig:subfig:a}`	65a
`\subref{fig:subfig:b}`	(b)
`\subref*{fig:subfig:b}`	b
`\ref{fig:subfig:b}`	65b
`\ref{fig:subfig}`	65

28.3.1 Minipage Environments Inside Subfigures

Since Subfigure 65a consists of only the `\includegraphics` command, the caption in subfigure 65a is only as wide as the included graphic. If the subfigure instead consists of the entire minipage, the caption is made as wide as the minipage. For example

```
\begin{figure}
    \subfloat[Small Box with a Long Caption]{
    \label{fig:mini:subfig:a}   %% label for first subfigure
    \begin{minipage}[b]{0.45\linewidth}
       \centering \includegraphics[width=1in]{graphic}
    \end{minipage}}%
    \hfill
    \subfloat[Big Box]{
    \label{fig:mini:subfig:b}   %% label for second subfigure
    \begin{minipage}[b]{0.45\linewidth}
       \centering \includegraphics[width=1.5in]{graphic}
    \end{minipage}}
    \caption{Minipages Inside Subfigures}
    \label{fig:mini:subfig}     %% label for entire figure
\end{figure}
```

produces Figure 66, which contains subfigures 66a and 66b.

Since subfigure captions are (by default) as wide as the subfigure, the subfigure captions in Figure 66 are wider than those in Figure 65. This is because the Figure 65 subfigures contain only the graphics while the Figure 66 subfigures contain minipages of width `0.5\linewidth`.

(a) Small Box with a Long Caption (b) Big Box

Figure 66: Minipages Inside Subfigures

29 Separate Minipages for Captions

Section 28.2 on page 105 described how to construct side-by-side figures by placing the graphics command and `\caption` command together inside a `minipage` environment. This section describes how placing the graphics command and `\caption` command in separate `minipage` environments can provide better vertical alignment.

The `[t]` options for the side-by-side minipages in Figures 63 and 64 cause the graphic baselines to be aligned (see Section 11.4 on page 36). This works well for non-rotated graphics as it causes the tops of the captions to be aligned. However, this does not work well when the graphics bottoms are not aligned. For example,

```
\begin{figure}
   \centering
   %%----start of first figure----
   \begin{minipage}[t]{.4\linewidth}
      \centering
      \includegraphics[width=2cm]{graphic}
      \caption{Box with a Long Caption}
   \end{minipage}%
   \hspace{1cm}%
   %%----start of second figure----
   \begin{minipage}[t]{.4\linewidth}
      \centering
      \includegraphics[width=2cm,angle=-30]{graphic}
      \caption{Rotated Box}
   \end{minipage}%
\end{figure}
```

produces Figures 67 and 68 which do not have their captions aligned. The `[b]` minipage options would not completely solve the problem, as it causes the bottom lines of the caption to be aligned.

Figure 67: Box with a Long Caption

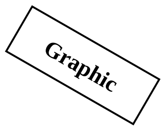


Figure 68: Rotated Box

The alignment of the graphics and the captions can be done separately by creating two rows of minipages: the first row containing the figures and the second row containing the captions. For example

```
\begin{figure}
   \centering
   %%----start of first figure graphics----
   \begin{minipage}[b]{.4\linewidth}
      \centering
      \includegraphics[width=2cm]{graphic}
   \end{minipage}%
   \hspace{1cm}%
```

```
    %%----start of second figure graphics----
    \begin{minipage}[b]{.4\linewidth}
       \centering
       \includegraphics[width=2cm,angle=-30]{graphic}
    \end{minipage}\\[-10pt]
    %%----start of first figure caption----
    \begin{minipage}[t]{.4\linewidth}
       \caption{Box with a Long Caption}
    \end{minipage}%
    \hspace{1cm}%
    %%----start of second figure caption----
    \begin{minipage}[t]{.4\linewidth}
       \caption{Rotated Box}
    \end{minipage}%
 \end{figure}
```

produces Figures 69 and 70, which have the graphic baselines aligned and the caption top lines aligned.

Figure 69: Box with a Long Caption

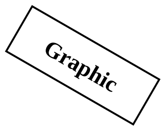


Figure 70: Rotated Box

Notes on this example

- The `\\` breaks the line after the last figure. The `\\` optional argument `[-10pt]` moves the captions closer to the graphics by removing 10 points of vertical space at the linebreak. This length should be changed as the user see fit.
- The graphic minipages have a `[b]` option to make their reference points be the baseline of the minipage's bottom line.
- The caption minipages have a `[t]` option to make their reference points be the baseline of the minipage's top line (to vertically-align the captions' top lines).
- Any `\label` commands must be issued in the same minipage as the corresponding `\caption` command.

30 Placing a Table Beside a Figure

In Section 28 on page 104, side-by-side figures are constructed by using multiple `\caption` commands in a single figure environment. Likewise, side-by-side tables are created by using multiple `\caption` commands in a single table environment.

The `\captionof` commands described in Section 21 on page 87 make it possible to put a table beside a figure. For example, the following commands

```
 \begin{figure}[htb]
    \begin{minipage}[b]{0.5\linewidth}
       \centering
       \includegraphics[width=0.8\linewidth]{graphic}
       \caption{This is a Figure by a Table}
       \label{fig:by:table}
    \end{minipage}%
    \begin{minipage}[b]{0.5\linewidth}
       \centering
       \begin{tabular}{|c|c|} \hline
```

```
         Day &  Data \\ \hline\hline
         Monday    & 14.6 \\
         Tuesday   & 14.3 \\
         Wednesday & 14.2 \\
         Thursday  & 14.5 \\
         Friday    & 14.9 \\ \hline
      \end{tabular}
      \captionof{table}{This is a Table by a Figure}
      \label{table:by:fig}
   \end{minipage}
\end{figure}
```

use a figure environment to create Figure 71 and Table 23.

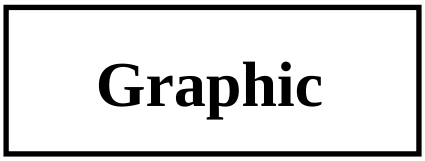


Figure 71: This is a Figure by a Table

Day	Data
Monday	14.6
Tuesday	14.3
Wednesday	14.2
Thursday	14.5
Friday	14.9

Table 23: This is a Table by a Figure

Since LaTeX allows figure floats to leapfrog table floats, using

```
\captionof{table}{...}
```

in a figure environment may place the table ahead of unprocessed tables. Likewise, using

```
\captionof{figure}{...}
```

in a table environment may place the figure ahead of unprocessed figures. If this is objectionable, it can be prevented by putting a `\FloatBarrier` or `\clearpage` command before the figure environment (see Section 17.3 on page 59).

31 Stacked Figures and Subfigures

Side-by-side figures are created in Section 28 on page 104 by a variety methods, all of which involve placing objects (graphics, minipages, subfloats) next to each other on a single line. The same procedure produces stacked graphics when the `\\` command is used to explicitly add a linebreak. The `\\` command's optional argument can specify additional vertical space, such as `\\[20pt]`.

31.1 Stacked Figures

Section 28 explained how to construct side-by-side figures. This section shows that adding a linebreak produces multiple rows of figures. For example, the following code

```
\begin{figure}[htbp]
   \centering
   %%----start of first figure----
   \begin{minipage}[t]{0.25\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
      \caption{First Stacked Figure}
      \label{fig:stacked:first}
   \end{minipage}%
```

```
   \hspace{1cm}%
   %%----start of second figure----
   \begin{minipage}[t]{0.25\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
      \caption{Second Stacked Figure}
      \label{fig:stacked:second}
   \end{minipage}\\[20pt]
   %%----start of third figure----
   \begin{minipage}[t]{0.25\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
      \caption{Third Stacked Figure}
      \label{fig:stacked:third}
   \end{minipage}%
   \hspace{1cm}%
   %%----start of fourth figure----
   \begin{minipage}[t]{0.25\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
      \caption{Fourth Stacked Figure}
      \label{fig:stacked:fourth}
   \end{minipage}%
   \hspace{1cm}%
   %%----start of fifth figure----
   \begin{minipage}[t]{0.25\linewidth}
      \centering
      \includegraphics[width=\linewidth]{graphic}
      \caption{Fifth Stacked Figure}
      \label{fig:stacked:fifth}
   \end{minipage}%
\end{figure}
```

produces Figures 72-76.

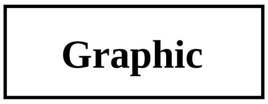


Figure 72: First Stacked Figure

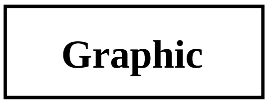


Figure 73: Second Stacked Figure

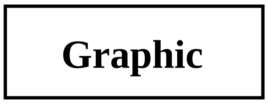


Figure 74: Third Stacked Figure

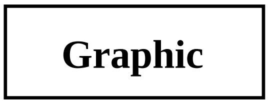


Figure 75: Fourth Stacked Figure

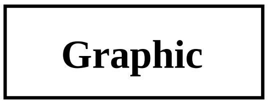


Figure 76: Fifth Stacked Figure

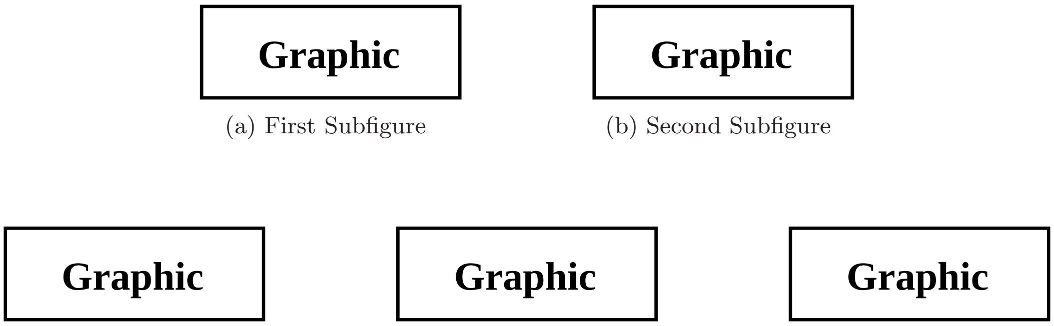
31.2 Stacked Subfigures

Section 28.3 on page 106 explained how to construct side-by-side subfigures. This section shows how adding a linebreak produces rows of subfigures. For example, the following code

```
\begin{figure}
   \centering
   %%----start of first subfigure----
   \subfloat[First Subfigure]{
```

```
          \label{fig:stacksub:a}        %% label for first subfigure
          \includegraphics[width=0.25\linewidth]{graphic}}
     \hspace{0.1\linewidth}
     %%----start of second subfigure----
     \subfloat[Second Subfigure]{
          \label{fig:stacksub:b}        %% label for second subfigure
          \includegraphics[width=0.25\linewidth]{graphic}}\\[20pt]
     %%----start of third subfigure----
     \subfloat[Third Subfigure]{
          \label{fig:stacksub:c}        %% label for third subfigure
          \includegraphics[width=0.25\linewidth]{graphic}}
     \hspace{0.1\linewidth}
     %%----start of fourth subfigure----
     \subfloat[Fourth Subfigure]{
          \label{fig:stacksub:d}        %% label for fourth subfigure
          \includegraphics[width=0.25\linewidth]{graphic}}
     \hspace{0.1\linewidth}
     %%----start of fifth subfigure----
     \subfloat[Fifth Subfigure]{
          \label{fig:stacksub:e}        %% label for fifth subfigure
          \includegraphics[width=0.25\linewidth]{graphic}}
     \caption{Five Subfigures}
     \label{fig:stacksub}               %% label for entire figure
  \end{figure}
```

produces Figure 77.



(a) First Subfigure (b) Second Subfigure

(c) Third Subfigure (d) Fourth Subfigure (e) Fifth Subfigure

Figure 77: Five Subfigures

32 The subfig package

This section provides an overview of the subfig package. Readers are encouraged to read Steven Douglas Cochran's subfig documentation [32] for the full details[41].

Section 28.3 provided an example that used the subfig package's `\subfloat` command to create subfigures. The example also used the `\label` command to define labels for the figure and the individual subfigures. This labels can be referenced using the `\ref` command and the subfig package's `\subref` and `\subref*` commands, with referencing examples described Table 22.

32.1 The Subfloat Command

The `\subfloat` command has one mandatory argument and two optional arguments. The mandatory argument contains the commands (such as `\includegraphics`) that generate the subfigure contents. The two optional arguments affect the subfigure's caption:

- If no optional arguments are specified then the subfigure has no subcaption
- If one optional argument is specified, its contents provide the subfigure's subcaption and its list-of-figures text.
- If two optional arguments are specified, the first (left) optional argument provides the subfigure's list-of-figure text while the second (right) optional argument provides its subcaption.

More detail on on the `\subfloat` command options are given in Table 24.

32.2 Customizing **subfig** with captionsetup Command

Section 20.1 on page 69 describes how Figure caption can be customized using the optional caption package. The same customization can be applied to subfigure captions created by the subfig package[42].

The `\captionsetup` command has an optional argument which specifies whether the customization applies to figures, subfigures, tables, subtables, or some combination. Table 25 shows the possible `\captionsetup` optional arguments and their effects.

[41] Since the subfig package requires the caption package, some of subfig package's capability comes from the caption code. Fortunately, the subfig documentation describes the full subfig capability, regardless of where the corresponding code resides.

[42] Since the subfig package automatically includes the caption package, the subfig always has all of the caption package's customization capabilities.

Table 24: `\subfloat` calling arguments

Command	List-of-Figures Caption	Sub-float caption
`\subfloat{body}`		
`\subfloat[]{body}`	(b)	(b)
`\subfloat[caption text]{body}`	(c) caption text	(c) caption text
`\subfloat[][caption text]{body}`		(d) caption text
`\subfloat[][]{body}`		(e)
`\subfloat[list text][caption text]{body}`	(f) list text	(f) caption text
`\subfloat[list text][]{body}`	(g) list text	(g)

Table 25: subfig package's `\captionsetup` options

Command	Description
`\captionsetup{`⟨*options*⟩`}`	options apply to all captions
`\captionsetup[figure]{`⟨*options*⟩`}`	options apply only to figures and subfigure captions
`\captionsetup[table]{`⟨*options*⟩`}`	options apply only to table and subtable captions
`\captionsetup[subfloat]{`⟨*options*⟩`}`	options apply only to all subfloats (subfigures and subtables)
`\captionsetup[subfigure]{`⟨*options*⟩`}`	options apply to subfigures
`\captionsetup[subtable]{`⟨*options*⟩`}`	options apply to subtables

In addition to the `\captionsetup` command's options listed on Page 73 - 75, there are some `\captionsetup` options which apply only to the `subfig` package, as shown in Table 26.

32.3 The ContinuedFloat Command

The `subfig` package also provides the `\ContinuedFloat` command which allows subfigures to be split between multiple figure environments (and thus multiple pages). This is useful when

- a figure has too many subfigures to fit on a single page
- or when an author wishes to relate multiple full-page graphics by having them numbered (17a, 17b, 17c) instead (18, 19, 20)

Examples of the `\ContinuedFloat` are provided in Section 33.

Table 26: subfig captionsetup Options

Keyword	Values	Default	Description
config=	<filename>	subfig.cfg	The filename from which to load subfig configuration.
lofdepth=	<integer>	1	If lofdepth=1, then only Figures are included in the List of Figures. If lofdepth=2, then both Figures and Subfigures are included in the List of Figures.
lotdepth=	<integer>	1	If lotdepth=1, then only Tables are included in the List of Tables. If lotdepth=2, then both Tables and Subtables are included in the List of Tables.
listofindent=	<length>	3.8em	Sets the total indentation from the left margin for List of Floats line for subfloats.
listofnumwidth=	<length>	2.5em	Sets the width of box for the label number for List of Floats line for subfloats.
farskip=	<length>	10pt	Vertical space on the "far side" of the subfloat (on the side away from the main caption).
nearskip=	<length>	0pt	Vertical space on the "near side" of the subfloat (on the side towards the main caption).
captionskip=	<length>	4pt	Vertical space between the subfloat and its subcaption
topadjust=	<length>	0pt	Extra vertical space added to captionskip when subcaption is above subfloat.
listofformat=	*(see Table 27)*	subparens	Specifies format of entries in List of Figures and List of Tables. Must be specified before \listoffigures
subrefformat=	*(see Table 27)*	subsimple	Specifies format of \subref* output. Format of \subref* output depends on the subrefformat= value when the subfloat is formed, regardless of the subrefformat= value when the \subref* command is entered.

Table 27: subfig captionsetup Options for listofformat= and subrefformat=

Values	Example	Description
subsimple	b	Only output subfigure letter.
subparens	(b)	Output subfigure letter surrounded by parens.
empty		Don't output anything.
simple	17b	Output figure number and subfigure letter.
parens	17(b)	Output figure number followed by subfigure letter surrounded by parens.

33 Continued Figures and Subfigures

When two successive figures contain closely-related material, it may be desirable to label the figures with the same figure number. Since the `figure` counter contains the number of the next figure, two figures can be given the same number by decrementing the `figure` counter before the figure environment. For example,

```
\begin{figure}
....
\end{figure}
\addtocounter{figure}{-1}
\begin{figure}
....
\end{figure}
```

However, the inability to distinguish between these identically-numbered figures causes confusion.

33.1 Continued Figures

The best way of constructing a continued figure is to use the subfig package to create multiple Figures, each of which contains a single subfigure. This allows the continued figures to be referenced individually as “Figure 12(a)” or collectively “Figure 12”. For example, the following code

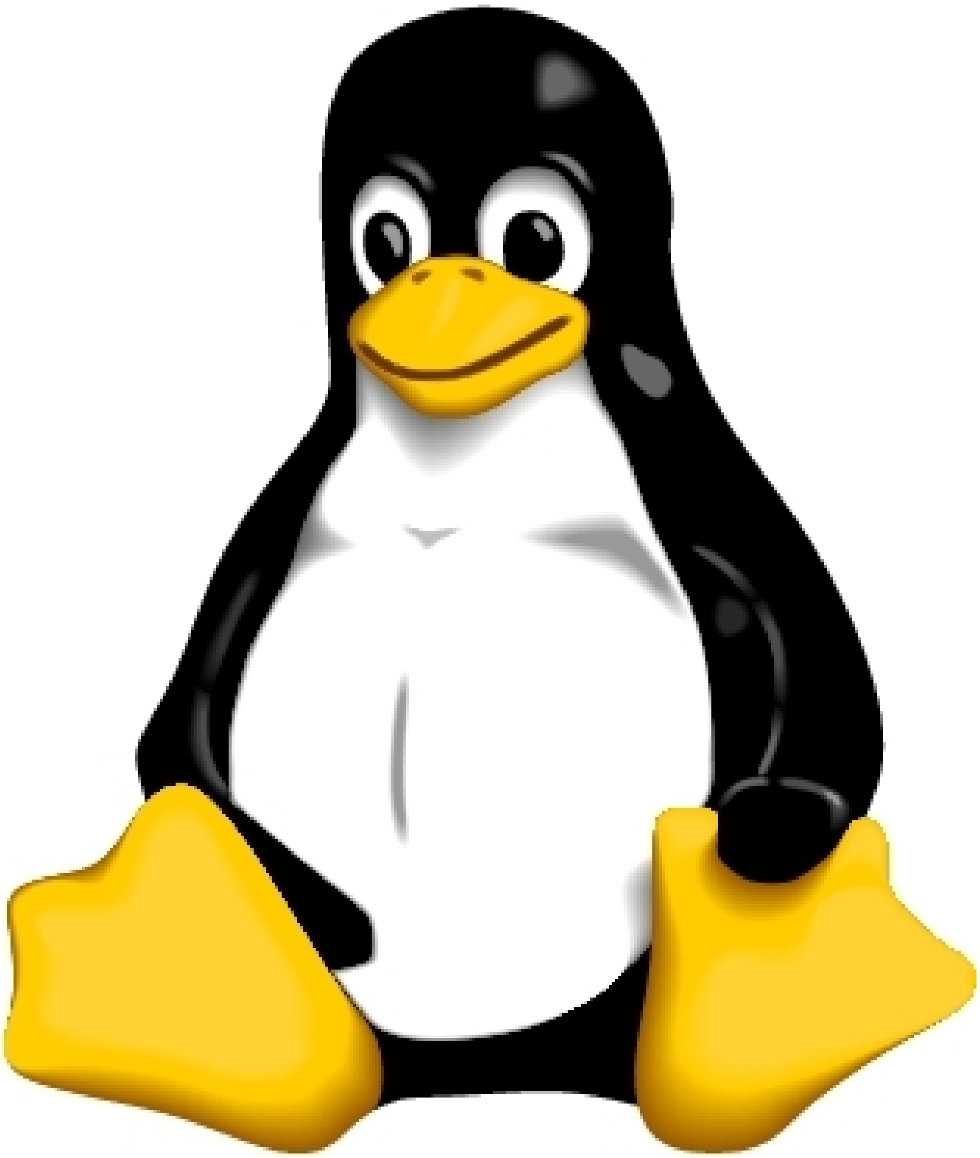
```
\usepackage{subfig,graphicx}
...
\begin{figure}[tbp]
   \centering
   \subfloat[Subcaption for First Part]{
        \label{subfig:continued:first}   %% label for first subfigure
        \includegraphics[height=6in]{tux}}
   \caption{Example of Continued Figure}
   \label{fig:continued:first}  %% label for first figure
\end{figure}

\begin{figure}[tbp]
   \ContinuedFloat
   \centering
   \subfloat[Subcaption for Second Part]{
        \label{subfig:continued:second}  %% label for second subfigure
        \includegraphics[height=6in]{tux}}
   \caption{Example of Continued Figure (cont'd)}
   \label{fig:continued:second}  %% label for second figure
\end{figure}
```

Produces Figure 78, which contains Figure 78a on Page 117 and Figure 78b on Page 118.

33.2 Continued Subfigures

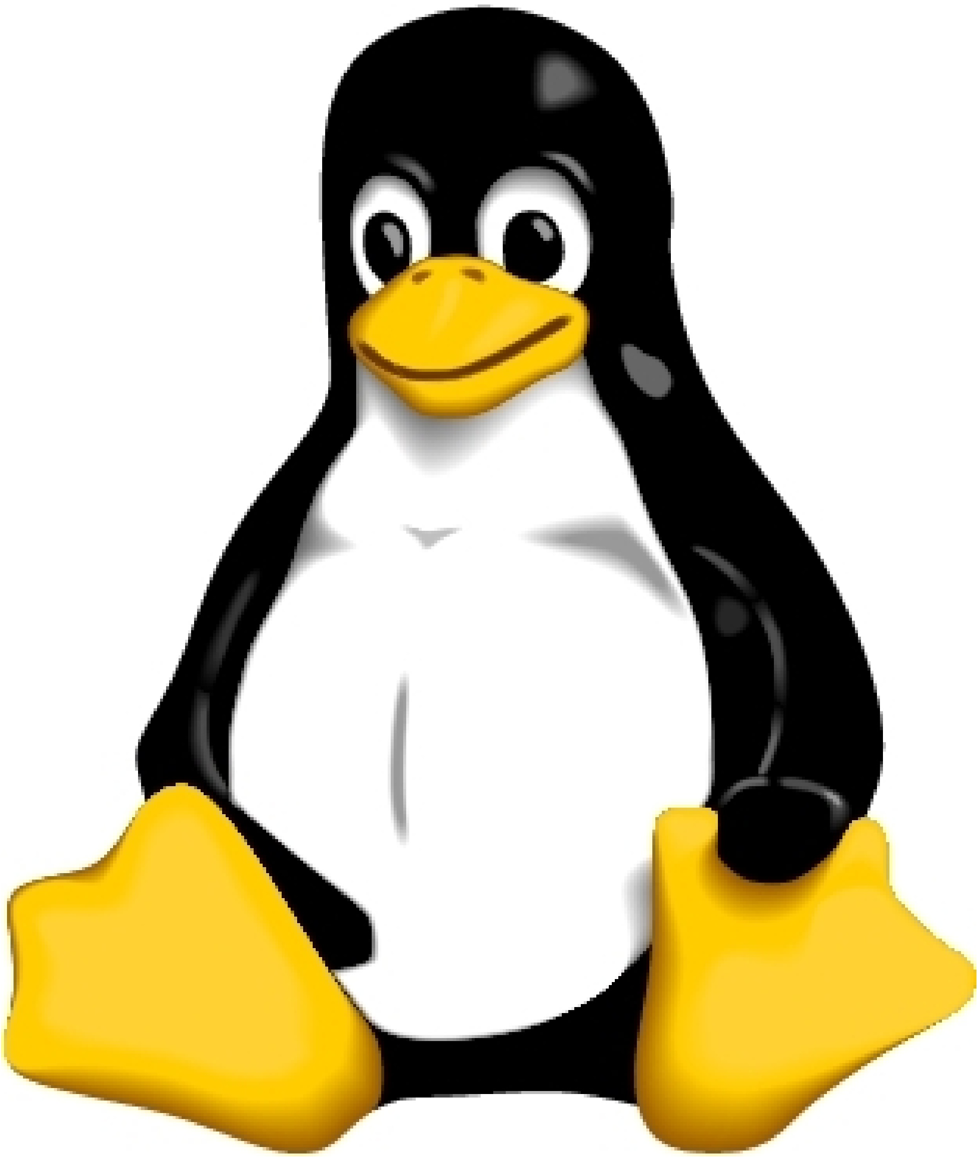
When grouping subfigures together in a figure, there often is not enough room to place all of the subfigures on a single page. Instead of breaking them into two differently-numbered Figures, the `\ContinuedFloat` command allows the two sets of subfigures to have the same Figure number. For example, the following code

```
\usepackage{subfig,graphicx}
...
\begin{figure}
   \centering
   %%----start of first subfigure----
```



(a) Subcaption for First Part

Figure 78: Example of Continued Figure



(b) Subcaption for Second Part

Figure 78: Example of Continued Figure (cont'd)

(a) First Subfigure (b) Second Subfigure

Figure 79: Two Subfigures

```
   \subfloat[First Subfigure]{
        \label{fig:contfig:subone}        %% label for first subfigure
        \includegraphics[width=3cm]{graphic}}
   \hspace{1cm}
   %%----start of second subfigure----
   \subfloat[Second Subfigure]{
        \label{fig:contfig:subtwo}        %% label for second subfigure
        \includegraphics[width=3cm]{graphic}}
   \caption{Two Subfigures}
   \label{fig:contfig:one}           %% label for first part
\end{figure}

\begin{figure}
   \ContinuedFloat
   \centering
   %%----start of third subfigure----
   \subfloat[Third Subfigure]{
        \label{fig:contfig:subthree}        %% label for third subfigure
        \includegraphics[width=3cm]{graphic}}
   \hspace{1cm}
   %%----start of fourth subfigure----
   \subfloat[Fourth Subfigure]{
        \label{fig:contfig:subfour}        %% label for fourth subfigure
        \includegraphics[width=3cm]{graphic}}
   \caption{Two Additional Subfigures}
   \label{fig:contfig:two}           %% label for second part
\end{figure}
```

Creates one float that contains Figures 79a and 79b and another float that contains Figures 79c and 79d. Note that the `\ContinuedFloat` command not only gives the floats the same Figure number, it also ensures that the second float's subfigure lettering does not reset to (a).

The two captions have different labels (`{fig:contfig:one}` and `{fig:contfig:two}`) which both produce the same `\ref` value but may produce different `\pageref` values since they can be on different pages.

Since four subfigures could easily fit in one float, this example obviously does not require the `\ContinuedFloat` command. But this example is meant to show the procedure for larger collections of subfigures.

(c) Third Subfigure (d) Fourth Subfigure

Figure 79: Two Additional Subfigures

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System, Second Edition*, Addison-Wesley, Reading, Massachusetts, 1994, ISBN 0-201-52983-1

[2] Helmut Kopka and Patrick Daly, *A Guide to LaTeX, Fourth Edition*, Addison-Wesley, Reading, Massachusetts, 2004, ISBN 0-321-17385-6

[3] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, and Chris Rowley, *The LaTeX Companion, Second Edition*, Addison-Wesley Pearson Education, Boston, Massachusetts, 2004, ISBN 0-201-36299-6

[4] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, *The LaTeX Graphics Companion*, Addison-Wesley, Reading, Massachusetts, 1997, ISBN 0-201-85469-4

[5] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, *The LaTeX Web Companion*, Addison-Wesley, Reading, Massachusetts, 1999, ISBN 0-201-43311-7

[6] Hàn Thế Thành, Sebastian Rahtz, Hans Hagen, Hartmut Henkel, Pawel Jackowski, *The pdfTeX user manual*, Available as `CTAN/systems/pdftex/manual/pdftex-l.pdf`

[7] D. P. Carlisle, *Packages in the 'graphics' bundle* (Documents the graphics, graphicx, lscape, color packages), Available as `CTAN/macros/latex/required/graphics/grfguide.ps`

[8] Tobias Oetiker, *The Not So Short Introduction to LaTeX 2ε* Available at `CTAN/info/lshort/english/lshort.pdf` and `CTAN/info/lshort/english/lshort.ps`

[9] Harvey Greenberg, *A Simplified Introduction to LaTeX* Available at `CTAN/info/simplified-latex/simplified-intro.ps`

[10] David Carlisle, *The afterpage package*, Available as `CTAN/macros/latex/required/tools/afterpage.dtx`

[11] LaTeX3 Project Team, *The calc package*, Available as `CTAN/macros/latex/required/tools/calc.dtx`

[12] Axel Sommerfeldt, *Typesetting captions with the caption package*, Available as `CTAN/macros/latex/contrib/caption/caption.pdf`

[13] Peter R. Wilson, *The ccaption package*, Available as `CTAN/macros/latex/contrib/ccaption/ccaption.pdf`

[14] James Darrell McCauley and Jeff Goldberg, *The endfloat Package*, Available as `CTAN/macros/latex/contrib/endfloat/endfloat.pdf`

[15] Rolf Niepraschk *The eso-pic package*, Available as `CTAN/macros/latex/contrib/eso-pic/eso-pic.pdf`

[16] Timothy Van Zandt, *Documentation for fancybox.sty*, Available as `CTAN/macros/latex/contrib/fancybox/fancybox.doc`

[17] Piet van Oostrum, *Page layout in LaTeX*, Available as CTAN/macros/latex/contrib/fancyhdr/fancyhdr.pdf

[18] *The flafter package*, Available as CTAN/macros/latex/unpacked/flafter.sty

[19] Anselm Lingnau, *An Improved Environment for Floats*, Available as CTAN/macros/latex/contrib/float/float.dtx

[20] Hendri Adriaens *The graphicx-psmin package*, Available as CTAN/macros/latex/contrib/graphicx-psmin/graphicx-psmin.pdf

[21] Sebastian Rahtz and Heiko Oberdiek *Hypertext marks in LaTeX: a manual for hyperref*, Available as CTAN/macros/latex/contrib/hyperref/doc/manual.pdf

[22] Heiko Oberdiek *The ifpdf package*, Available as CTAN/macros/latex/contrib/oberdiek/ifpdf.sty

[23] David Carlisle, *The ifthen package*, Available as CTAN/macros/latex/base/ifthen.dtx

[24] D. P. Carlisle, *The lscape package*, Available as CTAN/macros/latex/required/graphics/lscape.dtx

[25] John D. Hobby *A User's manual for MetaPost*, AT&T Bell Laboratories Computing Science Technical Report 162, 1992. Available as http://cm.bell-labs.com/who/hobby/cstr_162.pdf

[26] Don Hosek, *The morefloats package*, Available as CTAN/macros/latex/contrib/misc/morefloats.sty

[27] Rolf Niepraschk *The overpic package*, Available as CTAN/macros/latex/contrib/overpic/overpic.sty

[28] Donald Arseneau, *The placeins package*, Available as CTAN/macros/latex/contrib/placeins/placeins.sty

[29] Michael C. Grant and David Carlisle, *The PSfrag system, version 3*, Available as CTAN/macros/latex/contrib/psfrag/pfgguide.pdf

[30] Sebastian Rahtz and Leonor Barroca, *The rotating package*, Available as CTAN/macros/latex/contrib/rotating/rotating.dtx

[31] Rolf Niepraschk and Hubert Gäßlein *The sidecap package*, Available as CTAN/macros/latex/contrib/sidecap/sidecap.pdf

[32] Steven Douglas Cochran, *The subfig package*, Available as CTAN/macros/latex/contrib/subfig/subfig.pdf

[33] Robin Fairbairns *The topcapt package*, Available as CTAN/macros/latex/contrib/misc/topcapt.sty

[34] Frank Mittelbach *The varioref package*, Available as CTAN/macros/latex/required/tools/varioref.dtx

[35] David Carlisle, *The xr package*, Available as CTAN/macros/latex/required/tools/xr.dtx

Index