Using Imported Graphics in LaTeX 2_ε

Keith Reckdahl
reckdahl@am-sun2.stanford.edu

Version 2.0
December 15, 1997

Summary

This document explains how to use imported graphics in LaTeX 2_ε documents. While reading the entire document is certainly worthwhile, most users should be able to locate the necessary information by browsing the table of contents on pages 3-5 or the index on pages 84-86.

While LaTeX can import virtually any graphics format, Encapsulated PostScript (EPS) is the easiest graphics format to import into LaTeX. For example, EPS files are inserted by specifying

```
\usepackage{graphicx}
```

in the document's preamble and then using the command

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

Optionally, the graphic can be scaled to a specified height or width

```
\includegraphics[height=4cm]{file.eps}
\includegraphics[width=3in]{file.eps}
```

Additionally, the `angle` option rotates the included graphic

```
\includegraphics[angle=45]{file.eps}
```

The `\includegraphics` command and the rest of the LaTeX 2_ε graphics bundle are covered in Part II of this document.

This document is divided into the following four parts

Part I: Background Information

This part provides historical information and describes basic LaTeX terminology. It also describes the Encapsulated PostScript (EPS) format, differences between EPS and PS files, and methods for converting non-EPS graphics to EPS.

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 (reference [5]).

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.
- How to form side-by-side graphics, side-by-side figures, and side-by-side subfigures.
- How to construct continued figures which can span multiple pages.

Where to Get this Document

This document is available in PostScript form as `CTAN/info/epslatex.ps` or in PDF form as `CTAN/info/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/`
Deutschland	`ftp://ftp.dante.de/tex-archive/`
Eastern U.S.	`ftp://tug2.cs.umb.edu/tex-archive/`
Western U.S.	`ftp://ftp.cdrom.com/pub/tex/ctan/`
Australia	`ftp://unsw.edu.au/tex-archive/`
Japan	`ftp://ftp.riken.go.jp/pub/tex-archive/`

A complete list of CTAN mirrors can be obtained from the `CTAN.sites` file at any CTAN site, or by fingering `ctan@ftp.dante.de`

Contents

Acknowledgements

I thank David Carlisle for providing me with a great deal of assistance. I also thank Donald Arseneau, Robin Fairbairns, Jim Hafner, Piet van Oostrum, and other contributors to the `comp.text.tex` newsgroup, whose posts provided much of the information for this document. Thanks also goes to the many people who provided me with valuable suggestions and bug reports for this document.

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 allows TeX and LaTeX to import any graphic format which is supported by the DVI program being used.

Since DVI files are often converted to PostScript, the best-supported imported-graphic format is 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 2ε. 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 [5].

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, the wrapped packaged should be used only 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 converters generally do not support direct inclusion of most non-EPS formats, these graphics must be converted to EPS for insertion into LaTeX documents. In many

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

cases, this graphics conversion can be performed on-the-fly by the DVI-to-PS converter. Section 6 describes graphics-conversion programs while Section 13 describes how to use non-EPS graphics in LaTeX.

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.

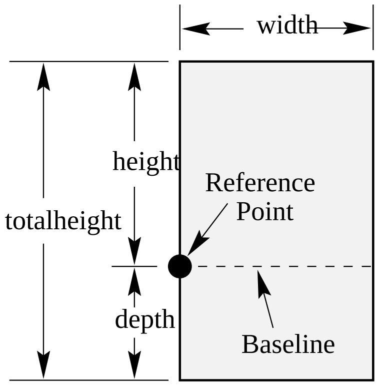


Figure 1: Sample LaTeX Box

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

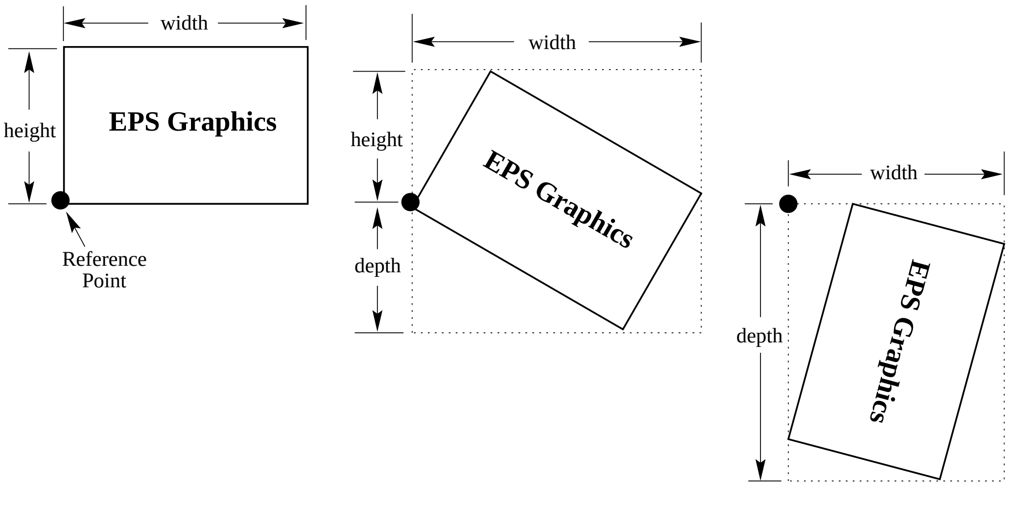


Figure 2: Rotated LaTeX Boxes

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.

3 Encapsulated PostScript

The PostScript language can describe 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 like misplaced or invisible graphics. Since many of these operators do not affect the PostScript stack, these problems can often be eliminated by simply deleting the offending operator. Other cases may require more complicated hacking of the PostScript.

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 5), 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 a 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. Alternatively, the BoundingBox parameters can be calculated and 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.

3.4 Fixing Non-standard EPS files

Some applications produce non-standard EPS files which cannot be used in other programs such as LaTeX. Some applications have developed their own "improved" flavor of PostScript with additional features, while other applications use poor PostScript programming. Fortunately, there are utilities which fix the non-standard EPS files created by the following applications.

Mathematica EPS files produced by Mathematica 2.x are written in Mathematica's extended PostScript. To use the EPS file in non-Mathematica programs, the file must be fixed to remove the non-standard extensions. DOS versions of Mathematica 2.x include a utility called either `printps.exe` or `rasterps` which removes the non-standard extensions. On Unix versions of Mathematica 2.x, this can be done with the `psfix` utility. Reference your Mathematica documentation or contact Wolfram Research for more information.

FrameMaker PostScript produced by FrameMaker fails to follow Adobe's specification for page independence. PostScript files produced by FrameMaker Version 4 and 5 can be respectively fixed using the scripts

```
ftp://ftp.irisa.fr/pub/FrameMaker/Filters/fixfm4-1.3.tar.gz
ftp://ftp.irisa.fr/pub/FrameMaker/Filters/fixfm5-2.0.tar.gz
```

Correction scripts for FrameMaker Version 3 and 4 are also available from

```
ftp://ftp.frame.com/pub/techsup/framers/platform.ind/filters/fixfm3ps.sh
ftp://ftp.frame.com/pub/techsup/framers/platform.ind/filters/fixfm4ps.sh
```

4 How EPS Files are Used by LaTeX

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

- LaTeX never even reads the EPS file if the BoundingBox parameters are specified in the graphics-insertion command (e.g., the `bb` option of `\includegraphics` is used).
- 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 13).
- 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 do not appear in most DVI viewers. To help the user with placement of the graphics, DVI viewers generally display the BoundingBox in which the graphics will be inserted. (Some viewers, such as `xdvi`, use ghostscript to interpret the EPS graphics and display them in the DVI viewer.)

4.1 Line Buffer Overflow

LaTeX reads ASCII files one line at a time, putting each line in its line buffer, which is usually 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 (this is not the case with `\psfig` and other obsolete graphics commands). 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 problem described in this section does not occur with the latest web2c distribution and some commercial distributions whose 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. However, such binary transfer results in incorrect end-of-line characters, requiring the BoundingBox information be provided by the graphics-insertion command (e.g., the `bb` option to `\includegraphics`).

5 Obtaining GhostScript

Ghostscript is a PostScript interpreter which runs on most platforms and is distributed for free[2] by Aladdin Enterprises. This allows PostScript and EPS files to be displayed on the screen and printed to non-PostScript printers. Aladdin Ghostscript is available from `CTAN/support/ghostscript/aladdin/`. It is also available directly from the Ghostscript home page

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

whose HTML interface provides better directions than do the CTAN FTP sites.

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 (GSview for Windows 3.1/95/NT/OS/2, Ghostview for Unix/VMS) for Ghostscript which makes the viewing of PostScript much easier.

[2] Although 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.

6 Graphics-Conversion Programs

The following freeware and shareware programs convert non-EPS graphics to EPS. Some of the programs allow command-line conversion which makes it possible to convert the graphics on-the-fly during `dvips` conversion (see Section 13.3).

- ImageMagick is a very good graphics-conversion utility that is distributed for free from `ftp.wizards.dupont.com` and other sites. See

  `http://www.wizards.dupont.com/cristy/ImageMagick.html`

  In addition to Unix and Linux, it now also runs under Windows NT, Macintosh, and VMS.
- `xv` is $25 shareware which provides graphics viewing and conversion programs for X-Windows systems. Note that `xv` does not provide command-line conversion capabilities for on-the-fly graphics conversion. For `xv` information, see

  `http://www.sun.com/sunsoft/catlink/xv/note.html`

  an on-line `xv` manual is available from

  `http://is.rice.edu/~shel/xv-3.10a/`
- `DISPLAY` is DOS freeware which converts between many types of graphic formats. It is available as `disp189a.zip` and `disp189b.zip` from the SimTel archives

  `http://www.simtel.net/simtel.net/msdos/graphics-pre.html`
  `http://oak.oakland.edu/simtel.net/msdos/graphics-pre.html`

  Future versions will increment the `189` version number.
- `WMF2EPS` is a freeware WMF-to-EPS conversion program which runs on Windows 95 and NT. It is available from

  `CTAN/support/wmf2eps/readme.txt`

  It requires an Adobe-compatible printer driver on your system.
- `KVEC` is $25 shareware which converts bitmap graphics (BMP, GIF, TIFF) into PostScript and other vector formats. `KVEC` is available for Windows, OS/2, NEXT and Unix.

  `http://ourworld.compuserve.com/homepages/kkuhl/`
- `NetPBM` is a maintained and improved version of the unsupported `PBMPLUS` package. It runs under Unix, VMS, and reportedly even DOS.

  `http://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM/`
- ImageCommander ($19 shareware) is a graphics-conversion program for Windows 3.1/95/NT which reads many types of graphics formats (GIF, JPEG, PICT, WMF, etc) and writes EPS and other formats. For more information, see

  `http://www.jasc.com/`

  JASC's Paint Shop Pro painting program ($69 shareware) has the same graphics-conversion capabilities.

6.1 Level 2 EPS Wrappers

Unlike conventional PostScript, Level 2 PostScript supports compressed binary graphics. 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.

- A JPEG graphic can be converted to level 2 PostScript by the C program `jpeg2ps` which is available from

  `http://www.muc.de/~tm/free/free.html`

  `jpeg2ps` can be compiled for Unix, DOS, and other systems.

- A TIFF graphic can be converted to LZW-encoded Level-2 PostScript by using `tiff2ps`, whose source is available from

  `ftp://ftp.sgi.com/graphics/tiff/tiff-v3.4-tar.gz`

  A `tar.Z` file is also available. `tiff2ps` can be compiled on Unix, DOS, Mac, and VMS platforms. Although the LZW PostScript files are small, they require a Level-2 PostScript printer.

6.2 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` is a free program for Unix, DOS, Windows, and OS/2 whose C++ source code is available from

  `ftp://ftp.x.org/contrib/applications/pstoedit/pstoedit.html`
  `http://www.cdrom.com/pub/X11/contrib/applications/pstoedit/`

  when used with ghostscript, `pstoedit` translates PostScript and PDF graphics into other vector formats (such as `xfig`'s `.fig` format).
- Mayura Draw (formerly known as PageDraw) is a drawing program for Windows 3.1/95/NT which is available from

  `http://www.wix.com/PageDraw`

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

  Older versions of Mayura Draw are distributed for free, while more recent versions are $15 shareware. Note that Mayura Draw requires Adobe Type Manager (ATM) to place text on the graphics. Although ATM is now commercial software, Adobe formerly distributed it for free with Acrobat Reader 2.0, which is available from Winsite at

  `ftp://ftp.winsite.com/pub/pc/win3/util/acroread.zip`
- `xfig` is a free drawing program for Unix/Xwindows available from

  `ftp://ftp.x.org/contrib/applications/drawing_tools/`
  `http://www.cdrom.com/pub/X11/contrib/applications/drawing_tools/`

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

Part II
The LaTeX Graphics Bundle

7 EPS Graphics Inclusion

The best reference for the graphics and graphicx packages is the graphics guide [5] or the *LaTeX Graphics Companion* [4]. The coverage of the graphicx package in other LaTeX references is sporadic: [2] covers both the graphics and graphicx packages, [1] only covers the graphics package and [3] describes neither.

7.1 The 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 . The commands

```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
   \includegraphics{file.eps}
\end{document}
```

include the graphics from `file.eps` at its natural size.

When the specified filename has no extension `\includegraphics` appends the extensions in the `\DeclareGraphicsExtensions` extension list (See Section 9.1). Since the default extension list does not include the null extension, `\includegraphics{file}` does *not* read `file` unless the null extension is added to the extension list.

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. *(Added 6/95)*
`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. *(Added 12/95)* The possible origin points are the same as those for the `\rotatebox` command in Section 8.3. 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.

Specifying Width

The command

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

includes the graphics from `file.eps` scaled such that its width is 3 inches. Instead of making the width be a fixed length such as 3 inches, making the width of function of `\textwidth` or `\em` makes a document more portable. For example, the command

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. *(Added 6/95)* 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. *(Added 6/95)* 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

`noclip`	(default) The entire graphic appears, even if portions appear outside the viewing area.
`clip`	When `clip` is specified, any graphics outside of the viewing area are clipped and do not appear.
`draft`	When `draft` is specified, the graphic's BoundingBox and filename are displayed in place of the graphic, making it faster to display and print the document. The `draft` package option `\usepackage[draft]{graphicx}` causes all the graphics in a document to be inserted in draft mode.
`final`	(default, unless `\usepackage[draft]{graphicx}` is specified.) `final` causes the graphics to be displayed, it is used to override `\usepackage[draft]{graphicx}`.
`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. *(Added 9/95)*

```
\includegraphics[width=\textwidth]{graphic.eps}
```

scales the included graphic such that it is as wide as the text. The command

```
\includegraphics[width=0.80\textwidth]{graphic.eps}
```

makes the included graphic 80% as wide as the text. When the calc package is used, the following command make the graphic width 2 inches less than the width of text

```
\includegraphics[width=\textwidth-2.0in]{graphic.eps}
```

(This requires graphicx version *12/95* or later).

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.eps}
\includegraphics[width=4in]{file.eps}
\includegraphics[angle=45]{file.eps}
```

produce the same three graphics as

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

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

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.

The standard LaTeX 2_ε arguments `\height`, `\width`, `\totalheight`, `\depth` can be used to refer to the original size of `argument`. So `\resizebox{2in}{\height}{argument}` makes `argument` keep its same height but have a width of 2 inches.

The `\resizebox*` command is identical to `\resizebox`, except the second argument specifies the `totalheight` of the object.

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.

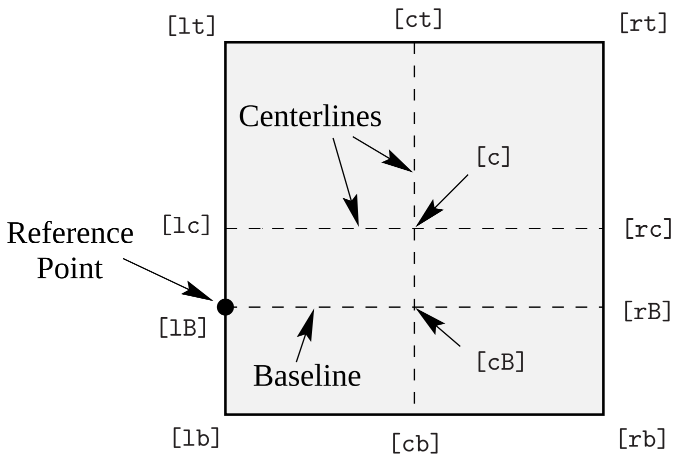


Figure 3: Available Origin Points

The horizontal position of the `origin` points is specified by one of three letters: `lcr` (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 Commands

This section describes advanced 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 (see Section 13.1).
3. When non-EPS graphics are used (see Section 13.3).

In these situations, the `\DeclareGraphicsRule` and `\DeclareGraphicsExtensions` commands are needed to control how LaTeX deals with 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[3]. For example if `dvips` is used, the following graphic extensions (defined in `dvips.def`) are used by default

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

With the above graphics extensions specified, `\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.

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.

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 12.3). If pool space is a concern, `\includegraphics{file}` should only be used with a `\DeclareGraphicsExtensions` command containing 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. Multiple `\DeclareGraphicsRule` commands may be issued.

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

For example, the command

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

[3] Specifying a graphics-driver package option such as `\usepackage[dvips]{graphicx}` overrides the default graphics driver specified in the `graphics.cfg` file.

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 forward single quote.) Currently, only `dvips` allows execution of such a command. See Section 13 for examples of using this command for use with compressed and non-EPS graphics.

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,

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

causes any unknown extension to be treated as an EPS file. For example, this causes `file.EPS`) to be treated as a 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[4]. For example if the `dvips` driver is used, the following graphic rules (defined in `dvips.def`) 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 bitmaps (see Section 13.3.2). The last command causes filenames with unknown extensions to be treated as an EPS file.

[4]Specifying a graphics-driver package option such as `\usepackage[dvips]{graphicx}` overrides the default graphics driver specified in the `graphics.cfg` file.

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.eps}
\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.eps}
\end{figure}
```

is similar to

```
\begin{figure}
\begin{center}
   \includegraphics[width=2in]{graphic.eps}
\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 18.1 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` 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.eps}
\includegraphics{file.eps}
```

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

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

puts no space between the graphics. When horizontal spacing is desired between graphics, the `\hspace` command inserts a specific amount of space[5] while `\hfill` inserts a rubber length which provides which expands to fill the available space. For example,

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

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

pushes the graphics to the left and right margins, while

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

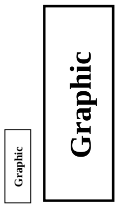
puts equal spacing before, between, and after the graphics. Since `\hfill` commands which occur before a linebreak are ignored, the `\hspace*{\fill}` was needed to supply the trailing space.

11 Rotation, Scaling, and Alignment

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.eps}
   \includegraphics[totalheight=1cm,angle=90]{graphic.eps}
\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.

11.1 Difference Between Height and Totalheight

Users must be careful when using the `height` option, as they often mean the overall height which is set by the `totalheight` option (see Figure 1 on Page 7). 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.eps}
\includegraphics[angle=-45,height=1in]{file.eps}
```

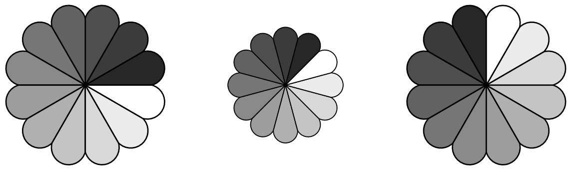
The first command scales the rotated graphic such that its total height is 1 inch. The second command scales the rotated graphics such that the portion above its reference point is 1 inch tall.

11.2 Scaling of Rotated Graphics

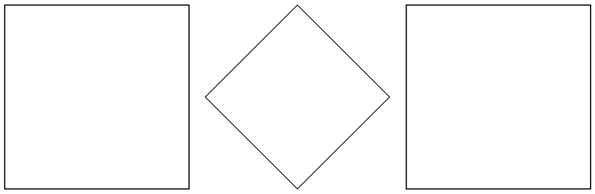
When the height or width of a graphic is specified, the specified size is not the size of the graphic but rather of its BoundingBox. This distinction is especially important in the scaling of rotated graphics. For example

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

produces



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



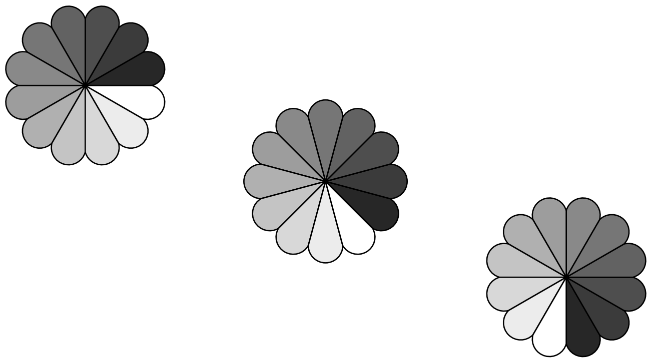
Each graphic is scaled such that its rotated BoundingBox is 1 inch tall.

11.3 Alignment of Rotated Graphics

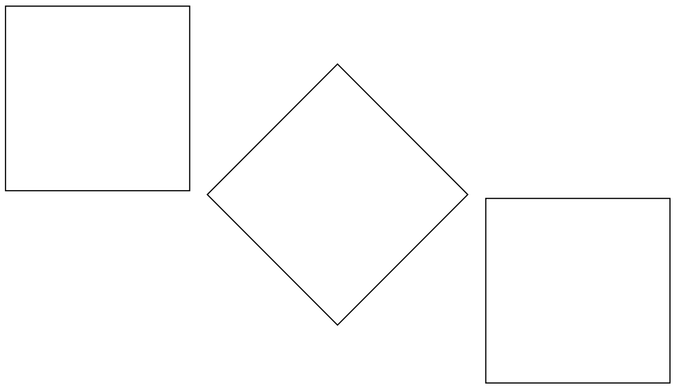
When graphics are rotated, the objects may not align properly. For example

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

produces



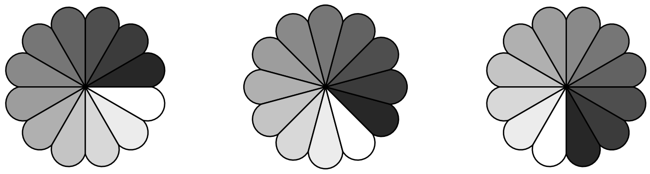
Again, this is better illustrated by the BoundingBoxes



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

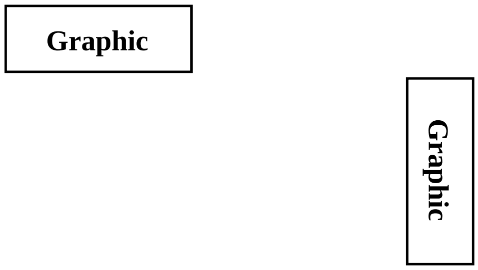
This aligns the centers of the graphics



Similarly, the commands

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

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



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

```
\begin{center}
   \includegraphics[width=1in]{graphic.eps}
   \hspace{1in}
   \includegraphics[width=1in,origin=br,angle=-90]{graphic.eps}
\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 27). 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 (unless the bottom line's reference point happens to be at its bottom.) Likewise, the `[t]` does *not* put the reference point at the top of the minipage (unless the top line's reference point happens to be at its top.)

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\textwidth}
      \centering
      \includegraphics[width=1in]{graphic.eps}
   \end{minipage}%
   \begin{minipage}[b]{.25\textwidth}
      \centering
      \includegraphics[width=1in,angle=-90]{graphic.eps}
   \end{minipage}
\end{center}
```

and

```
\begin{center}
   \begin{minipage}[t]{.25\textwidth}
      \centering
      \includegraphics[width=1in]{graphic.eps}
   \end{minipage}%
   \begin{minipage}[t]{.25\textwidth}
      \centering
      \includegraphics[width=1in,angle=-90]{graphic.eps}
   \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.

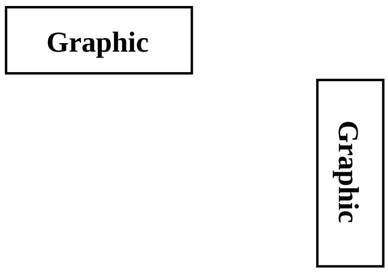


Figure 4: minipages with `[b]` or `[t]` options

11.4.1 Aligning the Bottoms of Minipages

One method for aligning the bottoms of minipages is to make the minipage baseline be the bottom of the minipage. If a line with zero height and zero depth is added inside the minipage after the graphics then the [b] option makes the bottom of the minipage be minipage's baseline. 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\textwidth}
      \centering
      \includegraphics[width=1in]{graphic.eps}
      \par\vspace{0pt}
   \end{minipage}%
   \begin{minipage}[b]{.25\textwidth}
      \centering
      \includegraphics[width=1in,angle=-90]{graphic.eps}
      \par\vspace{0pt}
   \end{minipage}
\end{center}
```

produces Figure 5.

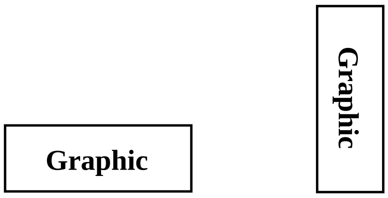


Figure 5: Minipages with Bottoms Aligned

11.4.2 Aligning the Tops of Minipages

To align the tops of the minipages, one must add a zero-height, zero-depth line to the top of the minipage. Then the [t] option makes the minipage baseline be the top of the minipage. Preceding \includegraphics command by \vspace{0pt} inserts a zero-height, zero-depth line above the graphic. 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\textwidth}
      \vspace{0pt}
      \centering
      \includegraphics[width=1in]{graphic.eps}
   \end{minipage}%
   \begin{minipage}[t]{.25\textwidth}
      \vspace{0pt}
      \centering
      \includegraphics[width=1in,angle=-90]{graphic.eps}
   \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 456-457].

Figure 6: Minipages with Tops Aligned

12 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 12.3). Since 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 12.1).
2. Another method is to specify `sub/` in a `\graphicspath` command (see Section 12.2). 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}
```

12.1 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 13.2).

12.2 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 12.3).

Due to these inefficiencies, it is recommended that the `\graphicspath` not be used. Instead, subdirectories should be specified by modifying the TeXsearch path (see Section 12.1).

12.3 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 solves 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), 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.

13 Compressed and Non-EPS Graphics Files

When using `dvips`, users can specify an operation to be performed on the file before it is inserted. 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.

When using an operating system which supports pipes[6], the `\DeclareGraphicsRule` (see Section 9.2) specifies a command which operates on the file. Making this command a decompression command allows compressed EPS files to be used. Making this command a graphics-conversion command allows non-EPS graphics files to be used. When using operating systems which do not support pipes, such on-the-fly conversion is not possible and all graphics must be stores as uncompressed EPS files.

13.1 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, the Unix command

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

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

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). 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}
```

13.2 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.

Old dvips Versions Since older versions of `dvips` do not search the TeX path, `dvips` cannot find files on the TeX path. The following command uses `kpsewhich` to search the TeX search path for non-compressed EPS files for `dvips`

```
\DeclareGraphicsRule{.eps}{eps}{.eps}{`cat `kpsewhich -n latex tex #1`}
```

(Although the better solution is to update your TeX distribution.)

13.3 Non-EPS Graphic Files

While it is easy to insert EPS graphics into LaTeX documents, it is not as straight-forward 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) 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) 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.

13.3.1 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}
```

13.3.2 Direct Support for Non-EPS Graphics

It is often requested that LaTeX and `dvips` support the direct inclusion of non-EPS graphic formats, making it as easy as inserting EPS files. While this would be convenient, there unfortunately are problems with this.

- To determine the size of EPS graphics, LaTeX scans the EPS file for the BoundingBox parameters. Since TeX can read only ASCII files, the binary format of most non-EPS graphic files prevents LaTeX cannot extract the graphic's size.
- Furthermore, supporting non-EPS graphics would require `dvips` to incorporate graphics-conversion capabilities (GIF-to-PS, TIFF-to-PS, etc.). This would require much initial programming and much future maintenance.

Rather than directly incorporating graphics-conversion routines, `dvips` provides a mechanism for calling external conversion programs. 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.

While LaTeX and `dvips` generally do not support the direct inclusion of non-EPS graphics, there are some exceptions.

1. If `dvips` is compiled with `-Demtex`, it supports some EmTeX `\special` commands, allowing it to include PCX, BMP, or MSP bitmaps.
2. Oztex 2.1, a shareware Macintosh TeX/LaTeX distribution, includes the DVI-to-PS converter OzDVIPS, which allows MacPaint and PICT files to be included via `\special` commands. See

   `http://www.kagi.com/authors/akt/oztex.html`
3. Some commercial versions of LaTeX support non-EPS graphics
   (a) Textures for the Macintosh supports PICT graphics. See
   `http://www.bluesky.com/`
   (b) Y&Y's TeX package for Windows includes the DVI-to-PS converter DVIPSONE which supports TIFF files. See
   `http://www.YandY.com/`

   Check your documentation or contact the company's customer service for the correct syntax.

Even with the above direct support for non-EPS, TeX can still not determine the size of graphics with binary-format files. In order for LaTeX to know how much space to allocate for the graphic, the user must still use a `.bb` file or specify the `bb` parameters explicitly in the `\includegraphics` command.

14 The PSfrag Package

While there are many drawing and analysis packages which produce EPS files, most of them 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.

PSfrag 3.0, which was released in November 1996, has been totally re-written. 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 (*12/95* or later) and the graphics bundle distributed with LaTeX. Reference [7] 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 14.3) 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.

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 counter-clockwise 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 `$x^2$`).

The `posn` and `PSposn` options are one of the 12 points (such as `[tl]`, `[br]`, `[cc]`) shown in Figure 3 on page 17. 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 [7] 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.

PSfrag cannot perform the replacement unless the entire EPS string is constructed with a single PS 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. If the kerning cannot be avoided, only single-character replacement strings can be used.

14.1 PSfrag Example #1

The commands

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

include the graphic without any PSfrag replacement, producing Figure 7. 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}
```

include the graphic with PSfrag replacement, producing Figure 8. 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 all EPS text need not be replaced. For example, the `N` tag is left unchanged in Figure 8.

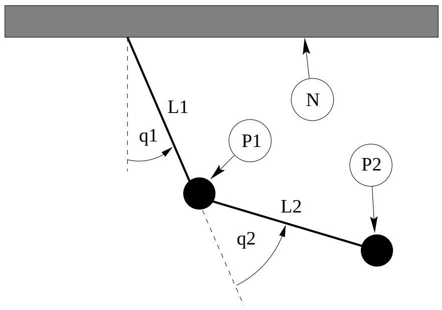


Figure 7: Without PSfrag Replacement

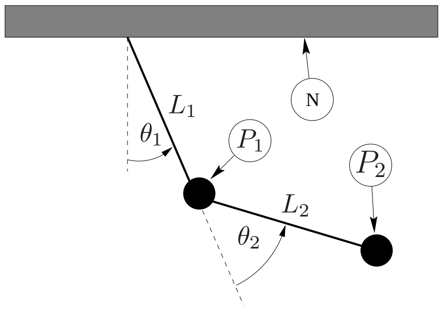


Figure 8: With PSfrag Replacement

14.2 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 reference [5] for more 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 9 and 10 demonstrate the use of these commands with PSfrag. Figure 9 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 10.

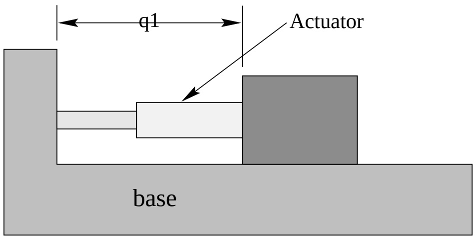


Figure 9: Without PSfrag Replacement

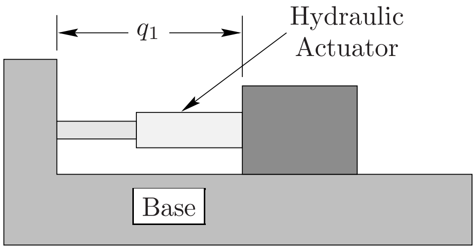


Figure 10: With PSfrag Replacement

14.3 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 [7] for more information.

14.4 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 [7] for more information on the scaling of PSfrag text.

14.5 PSfrag Incompatibilities

While PSfrag 3.0 has many advantages over version 2, it currently has an incompatibility with EPS files produced by Xfig which contain pattern-filled objects. The PSfrag distribution includes the file `readme.xfg` which describe this incompatibility. A work-around for this problem is described in Section 14.5.1.

The PSfrag distribution also includes the file `readme.sem` which describes an incompatibility between PSfrag and the Seminar package. Fortunately, the latest version of `seminar.cls` on CTAN no longer has this incompatibility.

14.5.1 Xfig EPS files

Problems occur when PSfrag is used with EPS files which are created with the xfig drawing program and use xfig's "pattern fill.' The problem stems from the fact that PSfrag and xfig both redefine PostScript's `show` command. The redefinitions shouldn't conflict with each other, but apparently they do.

The PSfrag maintainers haven't determined a long-term solution but the following work-around seems to work:

1. Inside the EPS file, look for the `/PATfill` command.
2. Inside the `/PATfill` definition, look for the `show` command (there is only one occurrence).
3. Replace `show` with `oldshow` (`oldshow` is the place where XFig stores the "old" version of the show routine, before it redefines "show" for its own purposes.)

If you can determine what xfig and/or PSfrag can do differently to avoid this problem, contact the PSfrag maintainers at `psfrag@rascals.stanford.edu`

15 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.

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}}
   ```

   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. The graphic can then be included by issuing the PostScript command wherever the graphic is needed. Section 15.1 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.

15.1 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
```

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.

15.2 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 [12]. 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 15.1, 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).

15.3 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\textwidth]{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 15.1.

Part IV

The Figure Environment

16 The Figure Environment

When using a word processor, figures appear exactly where the user places them. Since these figures cannot be split, they often lead to poor page breaks which 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.

To produce professional-looking documents without the figure-moving drudgery, LaTeX provides floating figures which automatically move to estheticly-pleasing locations. While these floating figures make it much easier to produce professional-looking documents, they often bother new users who are used to manual figure placement. Taking advantage of 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 should generally 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]` work well. See Section 16.2.
2. Many people find the default float parameters are too restrictive. The following commands

   ```
   \renewcommand{\textfraction}{0.15}
   \renewcommand{\topfraction}{0.85}
   \renewcommand{\bottomfraction}{0.65}
   \renewcommand{\floatpagefraction}{0.60}
   ```

   set the float parameters to more-permissive values. See Section 17.2.
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 should use the flafter package. No commands are necessary, simply include `\usepackage{flafter}`.
4. To guarantee that a figure does not float past a certain point, use the placeins package and issue a `\FloatBarrier` command. See Section 16.3.

   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.

16.1 Creating Floating Figures

Floating figures are created by putting commands in a `figure` environment. The contents of the figure environment always remains in one chunk, floating to 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. The `\label` command must be placed immediately *after* the `\caption` command.
- 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 27) or complex arrangements such as Figures 14-20 on page 53.
- A list of figures is generated by the `\listoffigures` command.
- 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 using long, descriptive captions.
- 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.

16.2 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[7].

`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]`.

[7]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.

- 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[8]. 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 logjam of figures exceeds LaTeX's limit of 18 unprocessed floats (see Section 16.4).

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[9]. 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.
   - 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 16.4.

   Similarly, a table cannot be placed until *all* previous tables are placed. However, tables can leapfrog figures and vice-versa.
5. The esthetic rules in Section 17 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 esthetic rules which apply to text pages (`!` does not affect the esthetic rules which apply to float pages).

16.3 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 17).

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.

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

[9] 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.

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

- The `\FloatBarrier` command causes all unprocessed floats to be processed immediately. Unlike `\clearpage`, it does not start a new page.
- 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.
- 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.

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 16.4). 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 17.2) 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.

16.4 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 17.2 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 of them. This is inconvenient because it requires some iterations to avoid partially-full pages. (Since
   ```
   \afterpage{\clearpage}
   ```
   causes a `\clearpage` at the next naturally-occurring pagebreak, it does not help in this situation because the float queue limit is reached before the pagebreak.)
   (c) Since there is no text, the figures don't need to float. Therefore, the best solution is to use the Section 20 procedure for constructing non-floating figures, separated by `\vspace` or `\vfill` commands to provide vertical spacing.

17 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.

17.1 Float Placement Counters

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).

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.

17.2 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}
```

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 60% of `\textheight`, they all cannot 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 40% 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.

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

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}{0.65}
\renewcommand{\floatpagefraction}{0.60}
```

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[10]. 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.

17.3 Suppressing Floats

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.

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.

18 Customizing the figure Environment

18.1 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}
```

[10]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 18.1 for information on figure spacing.

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.

The lengths listed in Table 9 do not affect the spacing of floats on float pages. Their spacing is controlled by the lengths listed in Table 10. The `fil` unit allows 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.

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 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`

The `@` in the names of the Table 10 lengths mean they are internal commands[11]. As a result, any `\setlength` command which modifies the lengths must be surrounded by `\makeatletter` and `\makeatother`. For example,

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

increases the space between floatpage floats by 4 points.

18.2 Horizontal Lines Above/Below Figure

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 18.1).
`\bottomfigrule`	This command is executed before the first float at the bottom of a page, but after the `\textfloatsep` spacing.

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

[11] 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, the `\makeatletter` command causes LaTeX to treat `@` as a letter, thus allowing `@` in command names. The `\makeatother` command causes LaTeX to revert to the normal behavior of treating `@` as a non-letter. Any user code which accesses or redefines internal commands must be surrounded by `\makeatletter` and `\makeatother`.

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 18.1).

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.
- The horizontal rules are as wide as the text, even if wider figures (see Section 22) 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.

18.3 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.eps}
\end{figure}
```

produce Figure 11, whose caption is placed quite close to the graphic.

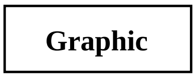
Figure 11: Caption Above Graphic

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.eps}
\end{figure}
```

produce Figure 12, which has has no extra space above the caption and 10 points of space between the caption and the graphic.

Figure 12: 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. 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.

18.4 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.eps}
   \renewcommand{\figurename}{Fig.}
   \caption{This is the Caption}
\end{figure}
```

produce Figure 13. The caption font, the ":" delimiter, and other caption characteristics can be customized with the caption2 package (see Section 19).

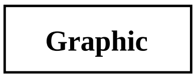


Fig. 13: This is the Caption

18.5 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 Customizing Captions with caption2

Sections 18.4 and 18.3 describe how to customize the caption label and caption vertical spacing. Other caption characteristics can be customized with the caption2 package[12].

The caption2 package can be used with many types of floats as it officially supports the float, longtable, and subfigure packages and it also works with the floatfig, rotating, supertabular, and wrapfig packages.

Syntax: `\usepackage[options]{caption2}`

Where the options are described in Table 12.

19.1 Caption Styles

The caption2 package defines the following caption styles, which are illustrated in in Figures 14-20.

normal Full lines are justified (aligned with both left and right margins) with the last line being left-justified.

center All lines of the caption are centered.

flushleft All lines of the caption are left-justified, leaving the right side ragged.

flushright All lines of the caption are right-justified, leaving the left side ragged.

centerlast All the lines are justified with the last line being centered.

[12]Since the original caption package had some bad side-effects (such as the requirement that it had to be loaded *after* other packages) it was totally re-written and renamed caption2. Although the caption2 is technically still a beta version, it is quite stable and performs well.

Table 12: caption2 Options

Caption Style	`normal`, `center`, `flushleft`, `flushright`, `centerlast`, `hang`, `indent`	Selects the caption style (see Section 19.1).
Caption Fontsize	`scriptsize`, `footnotesize`, `small`, `normalsize`, `large`, `Large`	Selects the fontsize for the caption label (e.g., "Figure 12:") and the caption text.
Caption Label Font Shape	`up`, `it`, `sl`, `sc`	Makes the caption label (e.g., "Figure 12:") have upright, italic, slanted, or small caps shape, respectively. **Does not affect caption text.**
Caption Label Font Series	`md`, `bf`	Makes the caption label (e.g., "Figure 12:") have a medium or boldface series font, respectively. **Does not affect caption text.**
Caption Label Font Family	`rm`, `sf`, `tt`	Makes the caption label (e.g., "Figure 12:") have roman, sans serif, or typewriter font, respectively. **Does not affect caption text.**
One-Line Caption Formatting	`oneline`, `nooneline`	Controls the formatting for one-line captions (see Section 19.3)

indent Same as "normal" style except that the second and subsequent lines are indented by the length `\captionindent`. Since `\captionindent` is zero by default, a command such as `\setlength{\captionindent}{1cm}` must be used to set the indentation.

hang Same as "normal" style except that the second and subsequent lines are indented by the width of the caption label (e.g., "Figure 12:").

Usually these styles are specified as `\usepackage` options such as

```
\usepackage[centerlast]{caption2}
```

which makes all the captions in the document have `centerlast` style.

19.2 Changing the Caption Style

The `\captionstyle` command changes the caption style. Placing the `\captionstyle` command inside an environment changes only those captions in that environment. For example, the commands

```
\begin{figure}
   \captionstyle{centerlast}
   \centering \includegraphics[width=3in]{graphic.eps}
   \caption{Centerlast Caption Style.  Centerlast Caption Style.}
\end{figure}
```

give only the current figure a `centerlast` style because `\captionstyle` is inside the figure environment. The commands

```
\captionstyle{centerlast}
\begin{figure}
   \centering \includegraphics[width=3in]{graphic.eps}
   \caption{Centerlast Caption Style.  Centerlast Caption Style.}
\end{figure}
```

give subsequent figures a `centerlast` style because `\captionstyle` is outside the figure environment.

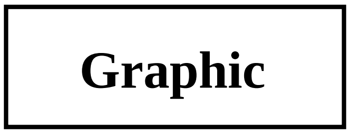


Figure 14: Normal Caption Style. Normal Caption Style. Normal Caption Style. Normal Caption Style.

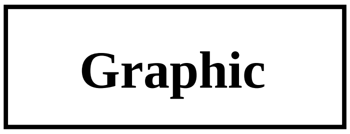


Figure 15: Center Caption Style. Center Caption Style. Center Caption Style. Center Caption Style.

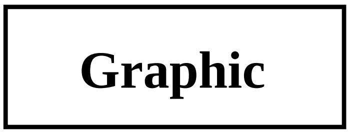


Figure 16: Centerlast Caption Style. Centerlast Caption Style. Centerlast Caption Style. Centerlast Caption Style.

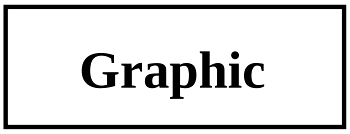


Figure 17: Flushleft Caption Style. Flushleft Caption Style. Flushleft Caption Style. Flushleft Caption Style.

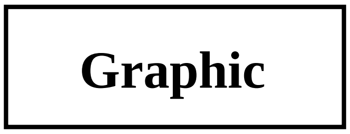


Figure 18: Flushright Caption Style. Flushright Caption Style. Flushright Caption Style. Flushright Caption Style.

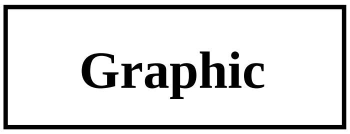


Figure 19: Indent Caption Style. Indent Caption Style. Indent Caption Style. Indent Caption Style.

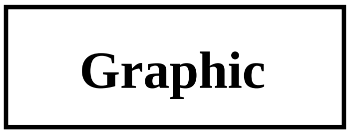


Figure 20: Hang Caption Style. Hang Caption Style. Hang Caption Style. Hang Caption Style.

19.3 One-Line Captions

If the caption is only one line, all of the above styles center the caption. To force the styles to be enforced even for one-line captions, one must include `nooneline` option

```
\usepackage[nooneline,flushleft]{caption2}
```

This formats *all* captions (including one-line captions) with the `flushleft` style. To change the `nooneline` option inside the document, `\onelinecaptionstrue` centers one-line captions while `\onelinecaptionsfalse` formats one-line captions. For example, the commands

```
\begin{figure}
   \captionstyle{flushleft}
   \onelinecaptionstrue
   \centering
   \includegraphics[width=2.5in]{graphic.eps}
   \caption{First Caption}
\end{figure}
```

center one-line captions as shown in Figure 21.
The commands

```
\begin{figure}
   \captionstyle{flushleft}
   \onelinecaptionsfalse
```

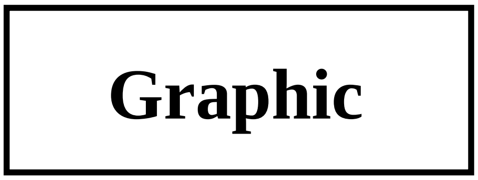


Figure 21: First Caption

```
    \centering
    \includegraphics[width=2.5in]{graphic.eps}
    \caption{Second Caption}
\end{figure}
```

causes one-line captions to be left-justified as shown in Figure 22

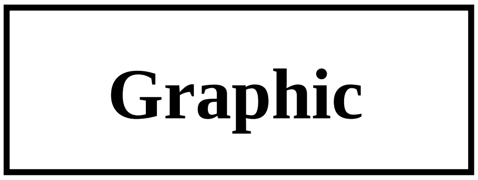


Figure 22: Second Caption

19.4 Caption Widths

The caption2 package provides functions which directly specify the captions' width/margins.

- `\setcaptionwidth{width}` sets the width of the caption to `width`, where `width` can be in any valid TeX units.
- `\setcaptionmargin{mar}` sets the margins to `mar`, making the caption width be the standard width minus 2 times `mar`.

  If `mar` is negative, the caption is made wider than the standard width, which is useful in subfigures and minipage environments.

For example, the commands

```
\begin{figure}
    \setcaptionwidth{2in}
    \centering
    \includegraphics[width=2in]{graphic.eps}
    \caption{Figure Caption Limited to Two Inches}
\end{figure}
```

make the caption 2 inches wide, as shown in Figure 23.

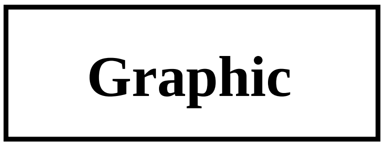


Figure 23: Figure Caption Limited to Two Inches

While the previous example directly set the width of the caption, alternatively the width can be indirectly set by specifying the spacing between the caption and each margin. For example, the commands

```
\begin{figure}
   \setcaptionmargin{1in}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{Figure Caption Where There is One Inch of
            Spacing between the Caption and Each Margin}
\end{figure}
```

indent both sides of the caption one inches from the page margins, as shown in Figure 24.

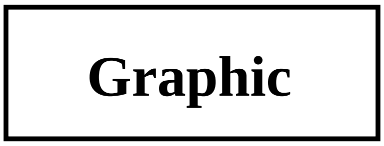


Figure 24: Figure Caption Where There is One Inch of Spacing between the Caption and Each Margin

19.4.1 Setting Caption Width to be Graphic Width

The previous section described how the `\setcaptionwidth` command gives the caption a specified width.

This section describes how to set the width of the caption to be the same as the width of the figure's graphic. This is very easy when the graphic is known

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

When the graphics width is unknown, the width can be determined by putting the graphic in a box and measuring the width of the box.

```
\newsavebox{\mybox}
\newlength{\mylength}
...
\begin{figure}
   \centering
   \sbox{\mybox}{\includegraphics[height=3in]{file.eps}}
   \settowidth{\mylength}{\usebox{\mybox}}
   \setcaptionwidth{\mylength}
   \usebox{\mybox}
   \caption{This is a figure with a very, very, very,
           very, very, very, very long caption}
\end{figure}
```

This can similarly be used with tables. `\mybox` and `\mylength` can be used multiple times within a document, but the `\newsavebox` and `\newlength` commands only need to be issued once.

19.5 Caption Delimiter

The default colon delimiter can be changed by redefining the `\captionlabeldelim` function. For example, the commands

```
\begin{figure}
   \renewcommand{\captionlabeldelim}{.}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{Caption with New Delimiter}
\end{figure}
```

change the delimiter in Figure 25 from the default colon to a period. If additional space is desired after the period,

```
\renewcommand{\captionlabeldelim}{.~}
```

can be used.

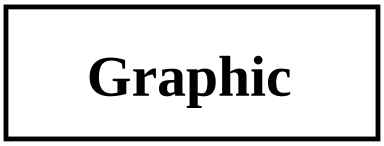


Figure 25. Caption with New Delimiter

19.6 Caption Font

While the `scriptsize,...,Large` options for `\usepackage{caption2}` change the size of both the caption label (e.g., "Figure 12:") and the caption text, the `up, it, sl, sc, md, bf, rm, sf, tt` options affect only the caption label.

The caption2 package also allows users to set the font of individual captions. The `\captionfont` sets the font for the caption label *and* the caption text, while `\captionlabelfont` sets the font for *only* the caption label. Thus, to set the font for only the caption text, `\captionfont` must be used to set the caption text font while `\captionlabelfont` must be used to set the label font, including "unsetting" any font properties set by `\captionfont`. The caption is effectively created by the following commands

```
{\captionfont%
    {\captionlabelfont \captionlabel \captionlabeldelim}%
    \captiontext}
```

where the `\captionlabel` command produces "Figure 12", the `\captionlabeldelim` command produces ":", and the `\captiontext` command produces the caption text.

LaTeX fonts are described by size and three type style components: shape, series, and family ([1, pages 37,115], [3, pages 170-71]). All four of these characteristics can be specified in the `\captionfont` and `\captionlabelfont` commands. For example, the commands

```
\begin{figure}
   \renewcommand{\captionfont}{\Large \bfseries \sffamily}
   \renewcommand{\captionlabelfont}{}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{Test Caption}
\end{figure}
```

produce Figure 26. In this example, the `\captionlabelfont` command does nothing. This means that it does not overwrite any font characteristics and all the `\captionfont` settings are carried over to the caption label. Since no shape declaration was specified, the entire caption has the default upright shape.

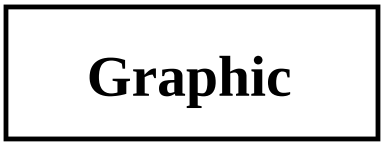


Figure 26: Test Caption

The commands

```
\begin{figure}
   \renewcommand{\captionfont}{\Large \bfseries \sffamily}
   \renewcommand{\captionlabelfont}{\small}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{Test Caption}
\end{figure}
```

produce Figure 27. In this example, the `\small` font size in `\captionlabelfont` overwrites the `\Large` font size from `\captionfont`. However, since `\captionlabelfont` does not contain any series or family declarations, the `\bfseries` and `\sffamily` declarations carry over to the caption label.

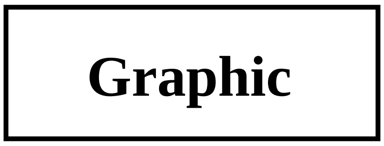


Figure 27: **Test Caption**

19.7 Custom Caption Styles

The caption2 package also allows users to create their own caption styles. For example, the following commands

```
\newcaptionstyle{one}{%
  \usecaptionmargin\captionfont%
  \onelinecaption%
    {{\bfseries\captionlabelfont\captionlabel\captionlabeldelim} \captiontext}%
    {{\centering\bfseries\captionlabelfont\captionlabel\par}\captiontext}}

\newcaptionstyle{two}{%
  \usecaptionmargin\captionfont%
    {\centering\bfseries\captionlabelfont\captionlabel\par}
    \onelinecaption{\captiontext}{\captiontext}}
```

define the captionstyles `one` and `two`. For captions of more than one line, both of these styles cause a bold caption label (e.g., **Figure 12**) placed on a separate line from the caption text. However, for short captions, style `two` puts the bold caption label on a separate line from the caption text, while style `one` puts them both on a single line separated by the delimiter. For example, after defining the above caption styles, the following code

```
\begin{figure}
   \captionstyle{one}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{First Custom Caption Style}
\end{figure}

\begin{figure}
   \captionstyle{two}
   \centering
   \includegraphics[width=2in]{graphic.eps}
   \caption{Second Custom Caption Style}
\end{figure}
```

produces Figures 28 and 29.

Notes about the custom caption styles

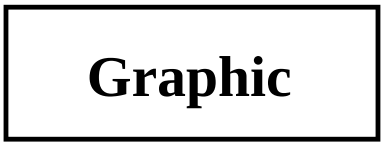


Figure 28: First Custom Caption Style

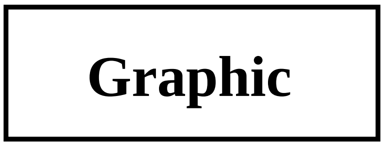


Figure 29
Second Custom Caption Style

- The `\onelinecaption` command takes two arguments: the first argument is performed if the caption is one line long, while the second argument is performed if the caption is more than one line long.
- When writing custom styles, use of commands such as `\captionfont` and `\captionlabelfont` are not required. However, their use is encouraged as it makes the styles more flexible.

  For example, the default `\bfseries` in the above custom style examples can be changed by setting `\captionlabelfont`. If such flexibility is not needed, the above custom style definitions could be shortened significantly.

19.8 Linebreaks in Captions

If a caption is longer than one line, linebreaks can be specified with `\protect\\`. When the caption fits in one line, it is processed in an `hbox`, which ignores any `\\` or `\par` .

The caption2 package allows linebreaks to be specified for captions of any length. For example, the commands

```
\begin{figure}
   \centering
   \includegraphics[width=3in]{graphic.eps}
   \captionstyle{center}
   \onelinecaptionsfalse
   \caption{First Line of Caption \protect\\ Second Line of Caption}
   \label{fig:caption:linebreak}
\end{figure}
```

produces the caption in Figure 30. Since `\\` is fragile[13], it must be preceded by `\protect`.

The `\onelinecaptionsfalse` (or `nooneline` package option) prevents LaTeX from processing the caption in an `hbox` which would lose the linebreak.

19.9 Adjusting Caption Linespacing

To doublespace a document, include either

```
\linespread{1.6}
```

[13]Some commands (such as `\textbf`) do not place any data in auxiliary files. Commands which save data for later use (e.g., `\caption` saves the caption text for the list of figures) are said to have *moving arguments*. Commands which break when used inside a moving argument are called *fragile* while commands which do not break when used inside a moving argument are called *robust*.

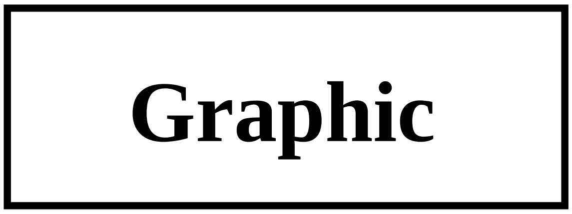


Figure 30: First Line of Caption
Second Line of Caption

or equivalently

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

in the preamble[14]. 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[15].

```
\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.

Regardless of whether `setspace` is used, the `caption2` package's `\captionfont` command can be used to adjust the caption spacing. For example,

```
\renewcommand{\captionfont}{\linespread{1.6}\normalsize}
```

produces doublespaced captions, regardless of the document's text spacing.

20 Non-Floating Figures

As described in Section 16, 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[16]. 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.eps}
\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
```

[14] 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.

[15] 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.

[16] Since it may produce large sections of vertical whitespace, preventing figures from floating is generally considered poor typesetting style. Instead, better results are generally produced by using the figure environment's `[!ht]` optional argument.

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. The following commands

```
This is the text before the figure.
\\[\intextsep]
   \begin{minipage}{\textwidth}
       \centering
       \includegraphics[width=2in]{graphic.eps}%
       \figcaption{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 between 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 18.1) 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[17]. 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 16.3).
- The \figcaption and \tabcaption commands are also useful for creating marginal figures (Section 21), and creating a table beside a figure (Section 29).

20.1 The float Package's [H] Placement Option

The float package[18] adds an [H] option to the figure environment which produces a non-floating figure. To use the [H] option, include a \usepackage{float} command in the preamble and issue the \restylefloat command *before* the \begin{figure}[H] command is used (See [3, page 149]). However, the float package's [H] placement option has the following side-effects

1. When the [H] figure does not fit on a page, the figure is moved to the top of the next page. However, if there were any footnotes on the first page, they appear directly below the text instead of at the bottom of the page. The user must manually insert some space before the figure in order to push the footnotes to the bottom of the page.
2. The figure environment defined by the float package always places the figure caption below the figure environment. While this does not affect simple figures, it prevents captions above graphics as in Figure 11 on page 49 or the construction of side captions (such as Figure 35 on page 67) and other complex figure arrangements (such as Figures 14–20 on page 53).

As a result, the \figcaption command defined in Section 20 is generally a better method for constructing non-floating figures than is the float package's [H] placement option.

---

[17] In these situations, the Table of Figures lists the figures in order of appearance, not in numerical order.

[18] The float package allows users to define new types of floats, such as "Program" or "Algorithm." It also define boxed and ruled ruled float styles.

21 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 was 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 their first line is vertically aligned with the line of the text which contains the `\marginpar` command (specifically, the reference point of the top line of the marginal note is aligned with the current baseline).

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 will 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.

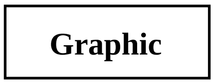


Figure 31: This is a Marginal Figure

Since the `figure` environment cannot be used in a marginal note, floating marginal figures are not possible. However, the `\figcaption` command defined in Section 20 can be used to construct a non-floating marginal figure. For example, Figure 31 was produced by

```
...to construct a non-floating marginal figure.
\marginpar{\centering
          \includegraphics[width=\marginparwidth]{graphic.eps}%
          \figcaption{This is a Marginal Figure}
          \label{fig:marginal:fig} }
For example, Figure~\ref{fig:marginal:fig} was...
```

The bottom of the graphic in Figure 31 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 the caption2 commands `\captionstyle{flushleft}` or `\captionstyle{flushright}` may provide better caption formatting. Additionally, the caption2 command

  ```
  \renewcommand{\captionfont}{\small}
  ```

  can be used to decrease the size of the caption font. See Section 19 for caption2 information.

- Like the non-floating figures in Section 20, 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 16.4).

22 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 21 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.

22.1 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 32.

```
\begin{figure}
\begin{narrow}{-1in}{0in}
   \includegraphics[width=\linewidth]{wide.eps}
   \caption{This is a wide figure}
\end{narrow}
\end{figure}
```

The specified width of `\linewidth` makes the graphic as wide as the `narrow` environment, while a width of `\textwidth` would make the graphic as wide as the original margins.

Figure 32: 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 `\textwidth` + `\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.

22.2 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.eps}
```

```
      \caption{Figure Caption}
      \label{fig:wide}
    \end{narrow}
  }% END ODD-PAGE FIGURE
  {% BEGIN EVEN-PAGE FIGURE
   \begin{narrow}{-1in}{0in}
      \includegraphics[width=\linewidth]{file.eps}
      \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.

23 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[19].
- The full-page figure produced by Option 2 floats to provide better document formatting, while the figure produced by Option 1 cannot float[20].
- Since Options 1 and 3 use the `figure` environment, they can be used in conjunction with the endfloat package (see Section 18.5).

23.1 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.

[19]The `landscape` environment works very well with the longtable package to produce multiple-page landscape tables.

[20]Figures issued in the `landscape` environment can float within the landscape pages

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.eps}
    \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.

23.2 Sidewaysfigure Environment

The rotating package provides the sidewaysfigure environment which produces figures with landscape orientation[21]. For example

```
\begin{sidewaysfigure}
   \centering
   \includegraphics[width=4in]{graphic.eps}
   \caption{Sidewaysfigure Figure}
\end{sidewaysfigure}
```

produces Figure 33.

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.

23.3 Rotcaption Command

The methods in Sections 23.1 and 23.2 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}
    \includegraphics[angle=90,width=\textwidth]{graphic.eps}
```

[21] The rotating package also provides a sidewaystable environment for producing tables with landscape orientation.

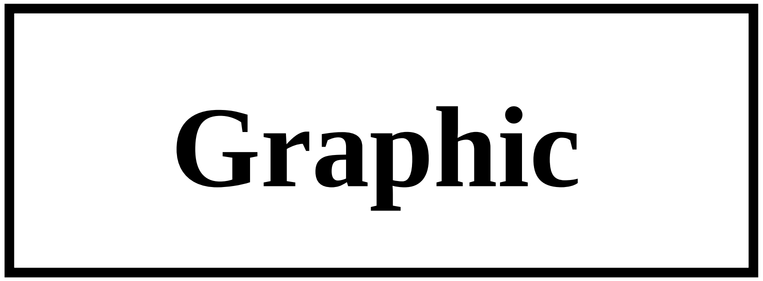


Figure 33: Sidewaysfigure Figure

```
      \end{minipage}
      \begin{minipage}[c]{0.5in}
         \rotcaption{Rotcaption Caption}
         \label{fig:rotcaption}
      \end{minipage}
   \end{figure}
```

produces Figure 34.

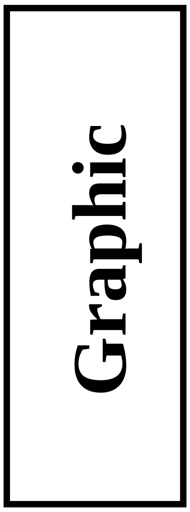


Figure 34: Rotcaption Caption

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 23.1 and 23.2, the `\rotcaption` command does not rotate the graphics. Therefore, the `\includegraphics` command in the above example requires the `angle=90` option.

24 Captions Beside Figures

Although the caption of a figure is generally place above or below the graphic, this section describes how to place the caption beside the graphic[22]. Section 24.1 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 24.2 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).

24.1 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

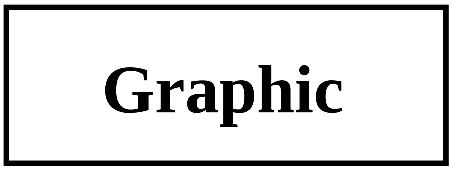
```
\begin{figure}
   \centering
   \begin{minipage}[c]{.45\textwidth}
      \centering
      \caption{Caption on the Side}
      \label{fig:side:caption}
  \end{minipage}%
```

[22]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 aspects of the float package can be used as long as the `\restylefloat` command is not issued.

```
    \begin{minipage}[c]{.45\textwidth}
        \centering
        \includegraphics[width=\textwidth]{graphic.eps}
    \end{minipage}
\end{figure}
```

produces Figure 35. It may be desirable to place a horizontal spacing command such as `\hfill` or `\hspace{.05\textwidth}` between the minipages.

Figure 35: Caption on the Side



The caption and graphic in Figure 35 are centered vertically. If it is instead desired to align the bottoms or tops of graphics and caption, see Section 11.4.

24.2 Caption on Binding Side of Graphic

The above code for Figure 35 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\textwidth}
         \centering
         \caption{Caption on the Side}
         \label{fig:side:caption}
      \end{minipage}%
      \hspace{0.05\textwidth}%
      \begin{minipage}[c]{.45\textwidth}
         \includegraphics[width=\textwidth]{graphic.eps}
      \end{minipage}%
   }% END ODD-PAGE FIGURE
   {% BEGIN EVEN-PAGE FIGURE
      \begin{minipage}[c]{.45\textwidth}
         \includegraphics[width=\textwidth]{graphic.eps}
      \end{minipage}%
      \hspace{0.05\textwidth}%
      \begin{minipage}[c]{.45\textwidth}
         \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.

24.3 The Sidecap Package

The methods in Sections 24.1 and 24.2 place caption to the side of figures. If one is willing to sacrifice some flexibility, the sidecap package can make the process easier.

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.eps}
   \caption{This is a SCfigure}
\end{SCfigure}
```

produces Figure 36.

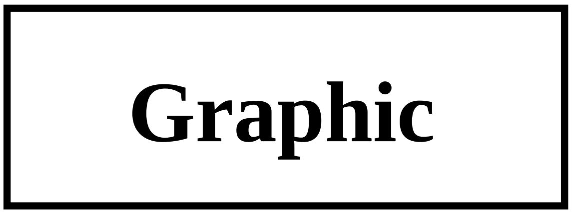


Figure 36: This is a SCfigure

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

- 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 16.2).

25 Figures on Even or Odd Pages

The figure environment float-placement algorithm does not control whether a figure appear on an even or odd page. To control odd/even placement, it is necessary to use the `\afterpage` command (part of the afterpage package) and the `\ifthenelse` command (part of the ifthen package) to move place a figure onto an odd or even page.

Putting the graphics in figure environments, might allow the even-page figure from floating to an odd page. Instead, the `\figcaption` command defined in Section 20 can be used to create a figure without using a figure environment.

```
\makeatletter
    \newcommand\figcaption{\def\@captype{figure}\caption}
\makeatother
```

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.eps}
    \figcaption{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}}
  ```

- Using `\value{page}` instead of `\pageref` is advantageous because `\value{page}` is always correct (`\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}{\textwidth}
             \centering
             \includegraphics{graphic.eps}
             \figcaption{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.eps}
      \figcaption{This is at the top of the left (even) page.}
      \vspace{\floatsep}}
  ```

25.1 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.eps}
        \figcaption{This is on the left (even) page.}
     \vspace*{\fill}\newpage\vspace*{\fill}%
        \centering
        \includegraphics{right.eps}
        \figcaption{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}}
```

26 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.

26.1 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.eps}}
   \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 37.

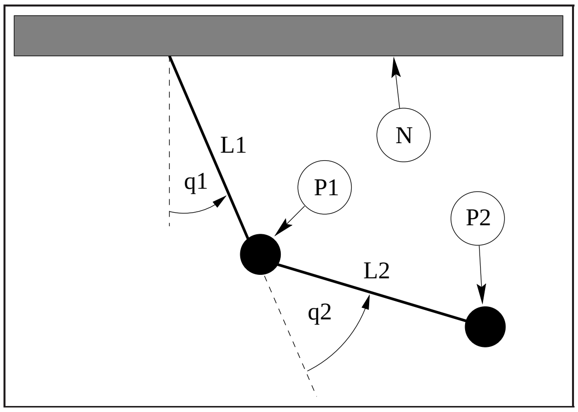


Figure 37: Box Around Graphic, But Not Around Caption

26.2 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[23].

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.eps}
          \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 38

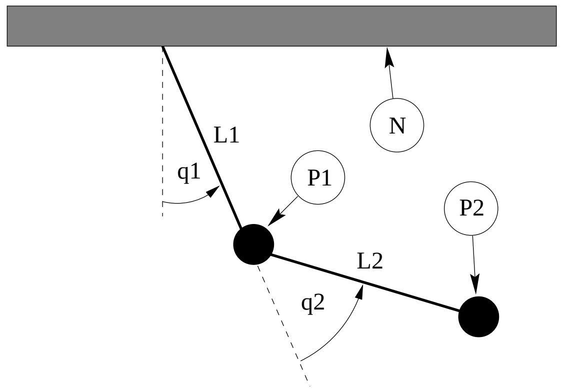


Figure 38: Box Around Figure Graphic and Caption

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=\textwidth]{pend.eps}
```

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.eps}}
\settowidth{\mylength}{\usebox{\mybox}}
\begin{figure}
```

[23] LaTeX uses three modes: LR mode, paragraph mode, and math mode. See [1, pages 36,103-5].

```
      \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}
...
```

26.3 Customizing fbox Parameters

In Figures 37 and 38, 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.eps}}
   \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 39

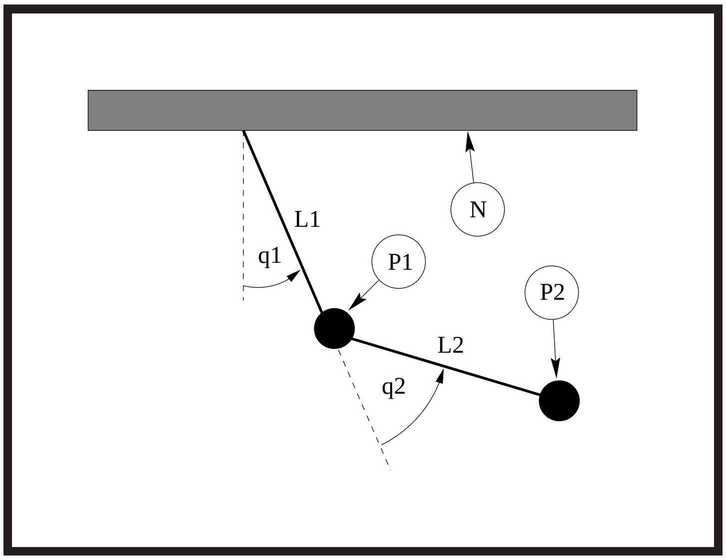


Figure 39: Graphic with Customized Box

26.4 The Fancybox Package

In Figures 37, 38, and 39, 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.

Table 13: 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`.

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` command, as was done for `\fboxrule` and `\fboxsep` in Section 26.3. 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.eps}
        \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 40.

27 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 43 and Figure 44).

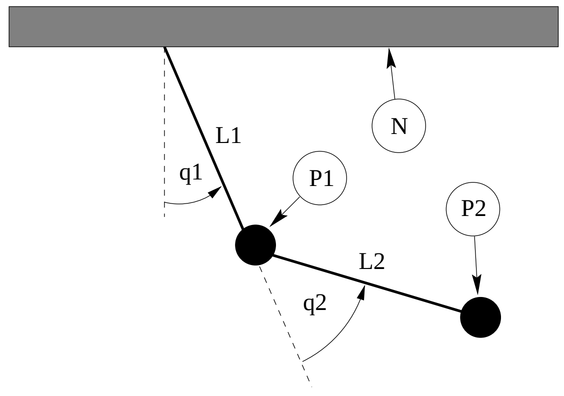


Figure 40: Shadowbox Around Entire Figure

3. The side-by-side graphics each form a subfigure (e.g., Subfigure 49(a) and Subfigure 49(b)) which are part of a single figure (Figure 49).

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 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.

27.1 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.

27.1.1 Using Side-by-Side includegraphics Commands

The following code

```
\begin{figure}
   \centering
   \includegraphics[width=1in]{graphic.eps}%
   \hspace{1in}%
   \includegraphics[width=2in]{graphic.eps}
   \caption{Two Graphics in One Figure}
\end{figure}
```

produces Figure 41 which is 4 inches wide (1 inch for `file1.eps`, 1 inch for the `\hspace`, and 2 inches for `file2.eps`) 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).

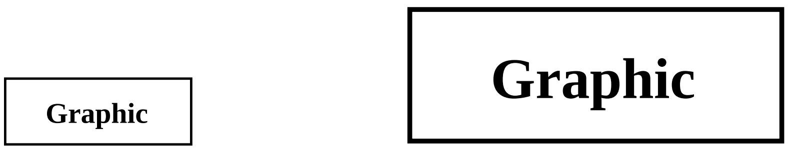


Figure 41: Two Graphics in One Figure

27.1.2 Using Side-by-Side Minipages

Placing the `\includegraphics` commands inside `minipage` environments provides the user more control over the graphics' vertical placement. For example

```
\begin{figure}
   \centering
   \begin{minipage}[c]{0.5\textwidth}
      \centering \includegraphics[width=1in]{graphic.eps}
   \end{minipage}%
   \begin{minipage}[c]{0.5\textwidth}
      \centering \includegraphics[width=2in]{graphic.eps}
   \end{minipage}
   \caption{Centers Aligned Vertically}
\end{figure}
```

produces Figure 42, which has vertically-centered graphics.

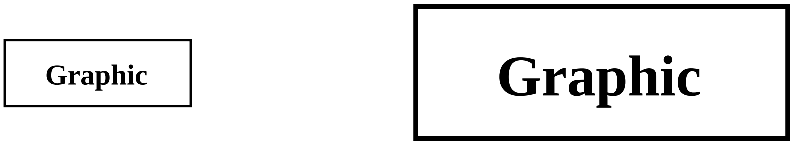


Figure 42: Centers Aligned Vertically

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).
- The `%` after the first `\end{minipage}` command prevents an interword space from being inserted between the `minipage` boxes (see Section 10.2).
- When the widths of the minipages do not add to 1.0`\textwidth`, the `\hspace` or `\hfill` commands can be used to specify horizontal spacing (see Section 10.2).

27.2 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}
   \begin{minipage}[t]{0.5\linewidth}
      \centering
      \includegraphics[width=1in]{graphic.eps}
      \caption{Small Box}  \label{fig:side:a}
   \end{minipage}%
   \begin{minipage}[t]{0.5\linewidth}
     \centering
     \includegraphics[width=1.5in]{graphic.eps}
     \caption{Big Box}  \label{fig:side:b}
   \end{minipage}
\end{figure}
```

produces Figures 43 and 44.

Although the above commands include *one* figure environment, the commands produce *two* figures because two `\caption` commands are used.

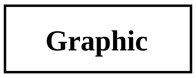


Figure 43: Small Box

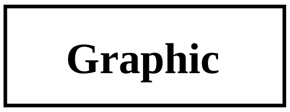


Figure 44: Big Box

27.2.1 Separate Minipages for Captions

The [t] options for the side-by-side minipages in Figures 43 and 44 cause the graphic baselines to be aligned (see Section 11.4). 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
   \begin{minipage}[t]{.33\textwidth}
      \centering
      \includegraphics[width=2cm]{graphic.eps}
      \caption{Box with a Long Caption}
   \end{minipage}%
   \begin{minipage}[t]{.33\textwidth}
      \centering
      \includegraphics[width=2cm,angle=-30]{graphic.eps}
      \caption{Rotated Box}
   \end{minipage}%
\end{figure}
```

produces Figures 45 and 46 which do not have their captions aligned. The [b] minipage options would not help, as it aligns the bottom lines of the caption.

Figure 45: Box with a Long Caption

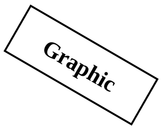


Figure 46: Rotated Box

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
   \begin{minipage}[b]{.33\textwidth}
      \centering
      \includegraphics[width=2cm]{graphic.eps}
   \end{minipage}%
   \begin{minipage}[b]{.33\textwidth}
      \centering
      \includegraphics[width=2cm,angle=-30]{graphic.eps}
   \end{minipage}\\[-10pt]
   \begin{minipage}[t]{.33\textwidth}
      \caption{Box with a Long Caption}
   \end{minipage}%
   \begin{minipage}[t]{.33\textwidth}
      \caption{Rotated Box}
   \end{minipage}%
\end{figure}
```

produces Figures 47 and 48, which have the graphic baselines aligned and the caption top lines aligned.

Figure 47: Box with a Long Caption

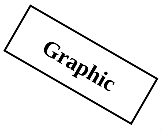


Figure 48: Rotated Box

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.
- Any `\label` commands must be issued in the same minipage as the corresponding `\caption` command.

27.3 Side-by-Side Subfigures

It may be desirable to refer to side-by-side graphics both individually and as a group. The `\subfigure` command (from the subfigure package) defines the group of side-by-side graphics as a single figure and defines each graphics as a subfigure. For example

```
\begin{figure}
   \centering
   \subfigure[Small Box with a Long Caption]{
        \label{fig:subfig:a}           %% label for first subfigure
        \includegraphics[width=1.0in]{graphic.eps}}
   \hspace{1in}
   \subfigure[Big Box]{
        \label{fig:subfig:b}           %% label for second subfigure
        \includegraphics[width=1.5in]{graphic.eps}}
   \caption{Two Subfigures}
   \label{fig:subfig}                  %% label for entire figure
\end{figure}
```

produces Figure 49. Typing `\ref{fig:subfig:a}` produces 49(a), typing `\ref{fig:subfig:b}` produces 49(b), and typing `\ref{fig:subfig}` produces 49.

(a) Small Box with a Long Caption

(b) Big Box

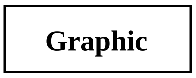
Figure 49: Two Subfigures

27.3.1 Minipage Environments Inside Subfigures

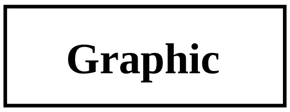
Like other side-by-side graphics, subfigures can be used with `minipage` environments. Depending on the situation, this may make it easier to achieve the desired spacing. For example

```
\begin{figure}
    \subfigure[Small Box with a Long Caption]{
    \label{fig:mini:subfig:a}   %% label for first subfigure
    \begin{minipage}[b]{0.5\textwidth}
       \centering \includegraphics[width=1in]{graphic.eps}
    \end{minipage}}%
    \subfigure[Big Box]{
    \label{fig:mini:subfig:b}   %% label for second subfigure
    \begin{minipage}[b]{0.5\textwidth}
       \centering \includegraphics[width=1.5in]{graphic.eps}
    \end{minipage}}
    \caption{Minipages Inside Subfigures}
    \label{fig:mini:subfig}     %% label for entire figure
\end{figure}
```

produces Figure 50, which contains subfigures 50(a) and 50(b).



(a) Small Box with a Long Caption



(b) Big Box

Figure 50: Minipages Inside Subfigures

Since subfigure captions are as wide as the subfigure, the subfigure captions in Figure 50 are wider than those in Figure 49. This is because the Figure 49 subfigures contain only the graphics while the Figure 50 subfigures contain minipages of width `0.5\textwidth`.

27.3.2 Changing Subfigure Numbering

The subfigure labels have two forms

1. One which appears under the subfigure as part of the caption. This is produced by the `\@thesubfigure` command.
2. One which appears when the `\ref` command is used. This is produced by concatenating the output of `\p@subfigure` to the output `\thesubfigure`.

These commands use the `subfigure` counter and the `\thefigure` command, making the subfigure label formating be controlled by the following commands

- The command `\thefigure` prints the current figure number.
- The counter `subfigure` counts the subfigures. The command `\alph{subfigure}` prints the value of the `subfigure` counter in lowercase letters, while `\roman{subfigure}` prints the value of the `subfigure` counter in lowercase Roman numerals. (see [1, page 98] or [3, page 446] for a list of counter output commands).
- The command `\thesubfigure` by default is `(\alph{subfigure})` which produces (a), (b), etc.
- The command `\@thesubfigure` by default is `\thesubfigure\space` which adds a space between the caption label and the caption.

- The command \p@subfigure by default is \thefigure

These commands make the default caption labels (a), (b), etc. and the default \ref labels 12(a), 12(b), etc. See [10] for controlling the size and font of the subfigure labels.

Subfigure Examples

1. To make the caption labels (i), (ii), etc. and make the \ref labels 12i, 12ii, etc. enter the following commands (preferably in the LaTeX file's preamble)

   ```
   \renewcommand{\thesubfigure}{\roman{subfigure}}
   \makeatletter
     \renewcommand{\@thesubfigure}{(\thesubfigure)\space}
     \renewcommand{\p@subfigure}{\thefigure}
   \makeatother
   ```

   The \makeatletter command tells LaTeX to treat the @ character as a letter, thus allowing the re-definitions of the internal commands. The \makeatother command tells LaTeX return to the normal behavior of treating the @ character as a non-letter.

2. To make the caption labels 12.1:, 12.2:, etc. and make the \ref labels 12.1, 12.2, etc. enter the following commands

   ```
   \renewcommand{\thesubfigure}{\thefigure.\arabic{subfigure}}
   \makeatletter
     \renewcommand{\@thesubfigure}{\thesubfigure:\space}
     \renewcommand{\p@subfigure}{}
   \makeatother
   ```

27.3.3 Adding Subfigures to List of Figures

By default, the List of Figures generated by the \listoffigures command includes only figures, *not* subfigures. To add the subfigures the List of Figures, type

```
\setcounter{lofdepth}{2}
```

before the \listoffigures command.

Note that a change in LaTeX has caused the current *3/95* subfigure package to add "numberline1" at the beginning of any subfigure entry in the List of Figures. To fix this, include the following code in the preamble of your document.

```
\makeatletter
\renewcommand{\@subcaption}[2]{%
   \begingroup
     \let\label\@gobble
     \def\protect{\string\string\string}%
     \xdef\@subfigcaptionlist{%
       \@subfigcaptionlist,%
       {\numberline {\@currentlabel}%
     \noexpand{\ignorespaces #2}}}%
   \endgroup
\@nameuse{@make#1caption}{\@nameuse{@the#1}}{#2}}
\makeatother
```

28 Stacked Graphics

In Section 27, side-by-side figures are arranged by placing blocks (either \includegraphics or minipages) beside each other on a line. Stacked graphics are built in exactly the same manner. For example,

```
\begin{figure}
   \centering
   \begin{minipage}[b]{0.3\textwidth}
      \centering
      \includegraphics[width=1in]{graphic.eps}
      \caption{Caption 1}
```

```
    \end{minipage}%
    \hspace{0.04\textwidth}%
    \begin{minipage}[b]{0.3\textwidth}
       \centering
       \includegraphics[width=1in]{graphic.eps}
       \caption{Caption 2}
    \end{minipage}\\[20pt]
    \begin{minipage}[b]{0.3\textwidth}
       \centering
       \includegraphics[width=1in]{graphic.eps}
       \caption{Caption 3}
    \end{minipage}%
    \hspace{0.04\linewidth}%
    \begin{minipage}[b]{0.3\textwidth}
       \centering
       \includegraphics[width=1in]{graphic.eps}
       \caption{Caption 4}
    \end{minipage}%
    \hspace{0.04\linewidth}%
    \begin{minipage}[b]{0.3\textwidth}
       \centering
       \includegraphics[width=1in]{graphic.eps}
       \caption{Caption 5}
    \end{minipage}
 \end{figure}
```

produces Figures 51-55. The `\\[20pt]` after the "Caption 2" minipage creates a line-break with 20 points of additional vertical spacing.

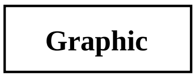


Figure 51: Caption 1

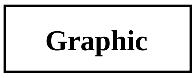


Figure 52: Caption 2

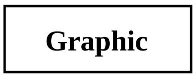


Figure 53: Caption 3

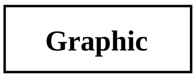


Figure 54: Caption 4

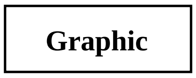


Figure 55: Caption 5

29 Placing a Table Beside a Figure

In Section 27, 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 `\figcaption` and `\tabcaption` commands defined in Section 20 make it possible to put a table beside a figure. For example, the following commands

```
\begin{figure}[htb]
   \begin{minipage}[b]{0.5\textwidth}
      \centering
      \includegraphics[width=0.8\textwidth]{graphic.eps}
      \caption{This is a Figure by a Table}
      \label{fig:by:table}
   \end{minipage}%
   \begin{minipage}[b]{0.5\textwidth}
      \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}
      \tabcaption{This is a Table by a Figure}
      \label{table:by:fig}
   \end{minipage}
\end{figure}
```

use a figure environment to create Figure 56 and Table 14.

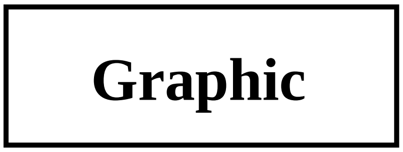


Figure 56: This is a Figure by a Table

Day	Data
Monday	14.6
Tuesday	14.3
Wednesday	14.2
Thursday	14.5
Friday	14.9

Table 14: This is a Table by a Figure

Since LaTeX allows figure floats to leapfrog table floats, using `\tabcaption` in a figure environment may place the table ahead of unprocessed tables. Likewise, using `\figcaption` in a table environment may place the figure ahead of unprocessed figures. If this is objectionable, it can be prevented by putting a `\FloatBarrier` command before the figure environment (see Section 16.3).

30 Continued Figures

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,

```
\addtocounter{figure}{-1}
\begin{figure}
```

However, the inability to distinguish between these similarly-numbered figures causes confusion.

The best way of constructing a continued figure is to use the subfigure package. This allows the continued figures to be referenced individually as “Figure 12(a)” or collectively “Figure 12”. Since the continued subfigures are in different figure environments, the `figure` counter must be decremented with

```
\addtocounter{figure}{-1}
```

between the figures, while the `subfigure` counter must be incremented with

```
\addtocounter{subfigure}{1}
```

at the beginning of the second figure environment. For example, the following code produces two continued subfigures.

```
\begin{figure}
   \centering
   \subfigure[First Part]{%
      \label{fig:graphics:a}% label for subfigure
      \includegraphics[width=\textwidth]{file1.eps}}%
   \caption{Large Graphics}%
   \label{fig:graphics}% label for figure
```

```
\end{figure}

\addtocounter{figure}{-1}
\begin{figure}
   \addtocounter{subfigure}{1}
   \centering
   \subfigure[Second Part]{%
       \label{fig:graphics:b}% label for subfigure
       \includegraphics[width=\textwidth]{file2.eps}}%
   \caption{Large Graphics (con't)}%
\end{figure}
```

In this example, each of the figure environments contain one subfigure. When each of the figure environments may contain multiple subfigures (such as in Section 27.3), the `subfigure` counter must be properly incremented to account for the subfigures in the first figure.

Since the continued figures are in separate floats, it is possible (but not likely) for them to not appear on successive pages. If this happens, the figures can be forced together by placing a `\FloatBarrier` command after the last continued figure.

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, Reading, Massachusetts, second edition, 1994, ISBN 0-201-52983-1

[2] Helmut Kopka and Patrick Daly, *A Guide to LaTeX 2ε*, Addison-Wesley, Reading, Massachusetts, 1995, ISBN 0-201-42777-X

[3] Michel Goossens, Frank Mittelbach and Alexander Samarin, *The LaTeX Companion*, Addison-Wesley, Reading, Massachusetts, 1994, ISBN 0-201-54199-8

[4] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, *The LaTeX Graphics Companion*, Addison-Wesley, Reading, Massachusetts, 1997, ISBN 0-201-85469-4

[5] D. P. Carlisle, *Packages in the 'graphics' bundle* (Documents the graphics, graphicx, lscape, color packages), Available as `CTAN/macros/latex/packages/graphics/grfguide.ps`

[6] Tobias Oetiker, *The Not So Short Introduction to LaTeX 2ε* Available as `CTAN/info/lshort/lshort2e.pdf` and `CTAN/info/lshort/lshort2e.600.ps`.

[7] Michael C. Grant and David Carlisle, *The PSfrag system, version 3*, Available as `CTAN/macros/latex/contrib/supported/psfrag/pfgguide.ps`

[8] David Carlisle, *The ifthen package*, Available as `CTAN/macros/latex/base/ifthen.dtx`

[9] David Carlisle, *The afterpage package*, Available as `CTAN/macros/latex/packages/tools/afterpage.dtx`

[10] Steven Douglas Cochran, *The subfigure package*, Available as `CTAN/macros/latex/contrib/supported/subfigure/subfigure.dtx`

[11] Harald Axel Sommerfeldt, *The caption package*, Available as `CTAN/macros/latex/contrib/supported/caption/caption2.dtx`

[12] Piet van Oostrum, *Page layout in LaTeX*, Available as `CTAN/macros/latex/contrib/supported/fancyhdr/fancyhdr.tex`

[13] Leonor Barroca, *The rotating package*, Available as `CTAN/macros/latex/contrib/supported/rotating/rotating.dtx`

[14] Timothy Van Zandt, *Documentation for fancybox.sty*, Available as `CTAN/graphics/pstricks/origdoc/fancybox.doc`

[15] Donald Arseneau, *The placeins package*, Available as `CTAN/macros/latex/contrib/other/misc/placeins.sty`

[16] *The flafter package*, Available as `CTAN/macros/latex/unpacked/flafter.sty`

[17] Don Hosek, *The morefloats package*, Available as `CTAN/macros/latex209/contrib/misc/morefloats.sty`

[18] James Darrell McCauley and Jeff Goldberg, *The endfloat Package*, Available as `CTAN/macros/latex/contrib/supported/endfloat/endfloat.dtx`

Index