The adjustbox Package

Martin Scharrer
martin@scharrer-online.de

CTAN: http://www.ctan.org/pkg/adjustbox

Version v0.8 – 2011/11/14

Abstract

This package got inspired by the `graphics/x` package and extends it sets of macros and generalises its key=value syntax for graphics to arbitray TEX material. Its main feature is the `\adjustbox` macro and the corresponding `adjustbox` environment which allow to adjust arbitrary TEX material in a multitude of ways, similar like `\includegraphics` does it for image files. In addition to this a wide set of dedicated macros are provided to modify a box. This startet with the two macros `\trimbox` and `\clipbox` which are missing in `graphics` but was extended over time.

The macros use the `collectbox` package to allow for verbatim content. Equivalent environments are also provided. The trim operation is now implemented in TEX and the clip operation uses `pdftex` primitives if available. Otherwise the `pgf` package is used for clipping, which supports both DVI/PS and PDF output. This package is still a little new and its implementation might not be fully stable yet.

1 Introduction

The standard LATEX package `graphicx` (the extended version of `graphics`) provides the macro `\includegraphics[`⟨*options*⟩`]{`⟨*file name*⟩`}` which can be used to include graphic files. Several options can be used to scale, resize, rotate, trim and/or clip the graphic. The macros `\scalebox`, `\resizebox` and `\rotatebox` are also provided to apply the corresponding operation on (LA)TEX material, which is subsequently placed inside a `\hbox`. However no macros are provided to trim or clip (LA)TEX material, most likely because this operations are not done by TEX but by the output format, i.e. using PostScript (PS) or PDF operations.

This package started by providing the missing macros `\clipbox` and `\trimbox`. Then a general `\adjustbox` macro which allows to combine many operations using a key=value syntax was added and further extended until it represented the main feature of this package. Newly added keys are also provided as dedicated macros and corresponding environments.

1.1 Driver dependent implementation

Certain operations, like clipping, depend on the output driver used, because they must be implemented as part of the output format, e.g. PDF instructions for a PDF

output file. The trim operation is now implemented as driver independent TEX code. The clip operation now uses pdftex primitives if possible, which are taken from the pdftex driver file of graphics. For other output formats a pgfpicture environment from the pgf package is used, which supports both PS and PDF output and comes which is own clipping drivers. All LATEX compilers should be supported, but pdflatex is the main target of the author.

1.2 Dependencies

The adjustbox package requires the author's other package collectbox, as well as the packages xkeyval, graphicx and ifpdf. The varwidth package is automatically loaded if installed, otherwise the varwidth and stack keys are disables as well as \stackbox/stackbox. For features which use colors the xcolor package must also be loaded manually (the color package is fine, too). The experimental split and pagebreak features also require the storebox package.

2 Package Option

Following v0.5 from 2011/08/13 this package accepts the following package options. Some of them can also be used as optional keys for macros.

minimal Only define the minimal set of macros, i.e. \trimbox, \clipbox and \adjustbox as with previous versions before v0.5.

export Export the now keys of \adjustbox also to \includegraphics so that they can be used for images as well. This option is meaningless if minimal was used.

pgf Uses the pgf package for all clip operations. This overrides all automatically detected drivers. At the moment only a pdftex driver is provided, all other compilers and output formats use this option already.

PGF Uses the pgf package for clip operations and configures the macros to parse lengths using pgfmath (compare with the pgf option below).

The following options define the way length values are processed by the provided macros. They can be used either as package options and as keys for \adjustbox (but not for \includegraphics even if the export option was used) to change the settings locally. The only difference between these two usages is that they also load required packages when used as package options. Therefore all keys used in the document should be loaded as package options first or the required packages must be loaded manually. (It is also possible to disable the advanced parsing of lengths using the none option, but this is not recommended.)

etex Uses the ϵ-TEX primitive \glueexpr to parse length values. This allows for additions, subtractions as well as multiplications and devision by a numeric factor. See the official etex_man document for more details. This setting is the default if ϵ-TEX is detected (which should be the case with all modern LATEX distributions).

calc Uses the calc package to parse length values. It supports all operations mentioned for etex and also some other operation like \widthof{⟨*text*⟩}. See the

`calc` package manual for more details. This is the default setting if ε-TeX is not detected.

pgfmath Uses the `pgfmath` package of the `pgf` bundle to parse length values. It supports all basic numeric operations and also advanced mathematical functions. See the `pgf` manual for more details. Because the `pgfmath` package can't be loaded independently in the current version (v2.10) the whole `pgf` package will be loaded.

One further option exists which can also be used as optional key for `\adjustbox` (but not for `\includegraphics`):

defaultunit =⟨*unit*⟩ This sets the default unit used for the values of `\trimbox`, `\clipbox` and `\marginbox` including there starred versions as well as all related keys like `trim`, `viewport`, `margin`, `trim`, `viewport`, `Clip` and `Clip*`. The standard default unit is the same as for `\includegraphics`: 'bp' (big points, PostScript points). However, for LaTeX material TeX normal unit 'pt' (TeX points) are better suited and will avoid rounding errors which otherwise get introduced by the internal conversion. The default unit is only used if the particular value is only a single number without unit, but not if any mathematical operations are used. If the special value `none` is used no default unit is applied and the internal check if the value is a single number is by-passed. This gives a small speed bonus and can be used to avoid potential issues with complex values. At this moment this setting will disable the default unit feature for the rest of the current group (i.e. all further `\adjustbox` keys or globally if used as a package option) and further usages of this option will have no affect. This might change in future versions of this package.

2.1 Verbatim Support

The macros read the content as a horizontal TeX box and not as an macro argument in order to support verbatim content. This functionality is now provided as dedicated package `collectbox` which can also be used independently. This means that the braces around the content can also be written as `\bgroup` and `\egroup`:

`\trimbox{1 2 3 4}\bgroup` ⟨*content*⟩`\egroup`

Special care is taken to allow the ⟨*text*⟩ to be a single macro (except `\bgroup`) without any braces:

```
\clipbox{1 2 3 4}\somemacro
```

This is to support the questionable habit of some LaTeX users to drop the braces for single token arguments. All environments support verbatim content.

3 Usage

This sections explains the existing features of this package. Most features are available as matching macro, environment and key.

Special care is taken so that the macros and the environments can have the same name. For starred environments the star can be either part of the name or an optional argument. Also the plainTeX syntax for environments (`\foobar ... \endfoobar`) can not be used because it will trigger `\foobar` as a macro.

3.1 Adjust Box Content

`\adjustbox{`⟨*key/value pairs*⟩`}{`⟨*content*⟩`}`

The `\adjustbox` macro is the general form of all box modifying macros mentioned in the introduction. It can be thought as an extended `\includegraphics` for (LA)TEX material. It supports the same ⟨*key/value pairs*⟩ and many more. However, they are provided as a mandatory not as an optional argument. An `\adjustbox` without options would not make sense and can be replaced by a simple `\mbox`. There is no starred version of this macro.

```
\begin{adjustbox}{⟨key=value, ...⟩}
  ⟨content⟩
\end{adjustbox}
```

The environment version of `\adjustbox`.

Adjust Images

`\adjustimage{`⟨*key/value pairs*⟩`}{`⟨*image filename*⟩`}`

This macro can be used as an extension of `\includegraphics`. While `\adjustbox` is based on the same interface as `\includegraphics` it provides more keys and allows global keys set by `\adjustboxset`. Most keys can be exported to `\includegraphics` using the export option, but there is no support for global keys[1]. Therefore it can make sense to use `\adjustbox{`⟨*key/value pairs*⟩`}{\includegraphics{`⟨*filename*⟩`}}`. The `\adjustimage` macro is a wrapper for this. However, it does not use `\includegraphics` directly, but an internal macro, to allow the redefinition of it. This does not require the export option and therefore helps to avoid option clashes if `adjustbox` is loaded at several places.

`\adjincludegraphics[`⟨*key/value pairs*⟩`]{`⟨*image filename*⟩`}`

Like `\adjustimage` but in the same format as `\includegraphics`. This macro allows to use all features of `\adjustbox` on images, however any '`[ ]`' inside the optional argument must be wrapped in '`{ }`' to mask them. It is possible to redefine `\includegraphics` to use/be `\adjincludegraphics` and this is done by the Export option (not to be confused with the export option).

3.1.1 Setting keys globally

[1] However some keys, but not all, can be set globally using `\setkeys{Gin}{`⟨*includegraphic key/value pairs*⟩`}`

`\adjustboxset{`⟨*global keys to be executed* before *local keys*⟩`}`
`\adjustboxset*{`⟨*global keys to be executed* after *local keys*⟩`}`

Using these two macros all keys can be set globally, i.e. for all future `\adjustbox` macros and `adjustbox` environments. Note that these settings are actually *local* to the current TEX group and only really global if used in the preamble or outside any group. The normal macro will place all given keys before the keys used in first argument of `\adjustbox` / `adjustbox`, while the starred version will place them afterwards.

If these macros are used several times there keys are accumulated. This happens in the given order for the normal version and in reversed order for the starred version, i.e. the keys of further `\adjustboxset` or `\adjustboxset*` are always added so they face inwards. If used without any keys but an empty argument, all keys previously set with the same macro are removed (from the current TEX scope). This means `\adjustboxset{}` clears all keys set be previously usages of `\adjustboxset{`⟨*keys*⟩`}` and `\adjustboxset*{}` clears all set by `\adjustboxset*{`⟨*keys*⟩`}`. Such resets are again local to the current TEX group.

Examples:

The macros:

```
\adjustboxset{keya=1}
\adjustboxset*{keyc=3}
\adjustbox{keyb=2}{content}
```

are effectively the same as:

```
\adjustbox{keya=1,keyb=2,keyc=3}{content}
```

The macros:

```
\adjustboxset{keya=1,keyb=2}
\adjustboxset{keyc=3,keyd=4}
\adjustboxset*{keyg=7,keyh=8}
\adjustboxset*{keyi=9,keyj=10}
\adjustbox{keye=5,keyf=6}{content}
```

are effectively the same as:

```
\adjustbox{keya=1,keyb=2,keyc=3,keyd=4,keye=5,keyf=6,
keyi=9,keyj=10,keyg=7,keyh=8}{content}
```

3.1.2 Argument Values

All length values given in the arguments of all macros and keys provided by this package are parsed by and advanced version of `\setlength(called \adjsetlength{)}` which uses either ϵ-TEX expressions (default), the `calc` package (default fall-back) or the `\pgfmathparse` of the `pgf` package. This allows for arithmetic expressions in these arguments. See the package options in section 2 to learn how to change the used length parser. Note that early versions of this package used `\pgfmathparse` by default. Older documents therefore might need now use the pgfmath option to compile correctly.

Parsing

Note that the four values for `\trimbox` and `\clipbox` as well as for the `trim` and `viewport` option of `\adjustbox` are separated by spaces. If the expression of any

Space=Separator

of this values holds a space or ends with a macro (eats trailing spaces!) it must be wrapped into braces '{ }'.

```
\width  \height  \depth  \totalheight
```

These LaTeX lengths hold the current dimensions of the content and can be used as part all length arguments. When the size of the content is changed by a key these lengths will be adjusted to hold the new size for all further keys. The totalheight is the height plus depth. With the `patch` option these lengths can also be used for `\includegraphics`.

```
\Width  \Height  \Depth  \Totalheight
```

These LaTeX lengths hold the original dimension of original unchanged content and are not modified. They are useful if the size of the content is modified by several keys, but further keys should still work relative to the original content.

Default unit

If no unit is provided for of the bounding box coordinates (llx, lly, urx, ury) in the trim and clip features then PostScript points (*big points*, bp, 72 bp = 1 inch) are used, as it is the default behaviour of the `trim` and `viewport` options of `graphicx`'s `\includegraphics`. Note that `graphicx` converts all values, independent if a unit is provided or not, internally to bp, because graphics where traditionally stored in Encapsulated PostScript (EPS) files. The more modern PDF files also use bp instead of pt. Because the `adjustbox` package macros target (La)TeX material and users will mostly use pt values this internal conversion to bp got disabled for them to avoid unnecessary rounding errors. Since v0.5 the default unit can be changed using the `defaultunit=`⟨*unit*⟩ key (which is also usable as global package option).

3.2 Trimming and Clipping

Material can be trimmed (the official size is made smaller, so the remaining material laps over the official boundaries) and clipped (overlapping material is not displayed). The original keys provided by `graphicx` are also mentioned here for comparison.

```
\trimbox{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
\trimbox{⟨all sites⟩}{⟨content⟩}
\trimbox{⟨left/right⟩ ⟨top/bottom⟩}{⟨content⟩}
\trimbox*{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
```

The macro `\trimbox` trims the given amount from the lower left (ll) and the upper right (ur) corner of the box. This means that the amount ⟨*llx*⟩ is trimmed from the left side, ⟨*lly*⟩ from the bottom and ⟨*urx*⟩ and ⟨*ury*⟩ from the right and top of the box, respectively. If only one value is given it will be used for all four sites. If only two values are given the first one will be used for the left and right side (llx, urx) and the second for the bottom and top side (lly, ury).

If the starred version is used the four coordinates are taken as the `viewport` instead, i.e. the box is trimmed to the rectangle described by the coordinates. In this case using all four values must be specified.

```
\begin{trimbox}{⟨1, 2 or 4 trim values⟩}
  ⟨content⟩
\end{trimbox}
```

```
\begin{trimbox*}{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}
  ⟨content⟩
\end{trimbox*}
```

The `trimbox` and `trimbox*` environments do the same as the corresponding macros.

```
trim=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
trim=⟨all sites⟩
trim=⟨left/right⟩ ⟨top/bottom⟩
```

This key represents the original `trim` key of `\includegraphics`. It always trims the original content independent from its position.

```
viewport=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
```

This key represents the original `viewport` key of `\includegraphics`. It always trims the original content to the given view port independent from its position.

```
Trim=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
Trim=⟨all sites⟩
Trim=⟨left/right⟩ ⟨top/bottom⟩
Viewport=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
```

The normal `trim` and `viewport` keys as described earlier are applied on the original content before any resizing or other effects. This is because for `\includegraphics` the trimming is done by the internal graphic driver, while the effects can be applied later (but can also be driver dependent). If the `trim` and `viewport` keys are used multiple times the last values will be used for the trimming, i.e. the content is only trimmed once. The upper case variants `trim` and `viewport` will wrap the content internally in a `\trimbox` or `\trimbox*` macro which can be applied multiple times, e.g. before and after the content is rotated. These two keys awaits the same format as the original keys. However, the `clip` key has no effect on them.

```
\clipbox{⟨all sites⟩}{⟨content⟩}
\clipbox{⟨left/right⟩ ⟨top/bottom⟩}{⟨content⟩}
\clipbox{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
\clipbox*{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
```

The `\clipbox` macro works like the `\trimbox` and trims the given amounts from the ⟨*text*⟩. However, in addition the trimmed material is also clipped, i.e. it is not shown in the final document. Note that the material will still be part of the output file but is simply not shown. It might be exported using special tools, so using `\clipbox` (or `\includegraphics[clip,trim=...]`) to censor classified information would be a bad idea. The starred version will again use the given coordinates as `viewport`.

```
\begin{clipbox}{⟨1, 2 or 4 trim values⟩}
  ⟨content⟩
\end{clipbox}
```

```
\begin{clipbox*}{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}
  ⟨content⟩
\end{clipbox*}
```

The environment versions of `\clipbox` and `\clipbox*`. The same rules as for the trimming environments apply.

`clip`

This boolean key represents the original `clip` key of `\includegraphics`. It is intended to be used to make `trim` or `viewport` clip the trimmed material.

```
Clip=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
Clip=⟨all sites⟩
Clip=⟨left/right⟩ ⟨top/bottom⟩
Clip*=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
```

As stated above the `clip` boolean key which will make the default `trim` and `viewport` keys clip the trimmed content, has no effect on the `trim` and `viewport` keys. Instead `Clip` and `Clip*` are provided which wrap the content internally in a `\clipbox` or `\clipbox*` macro. They can be used several times.

3.3 Margins

```
\marginbox{⟨all sites⟩}{⟨content⟩}
\marginbox{⟨left/right⟩ ⟨top/bottom⟩}{⟨content⟩}
\marginbox{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
```

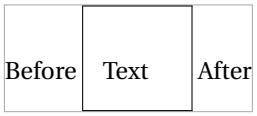
```
\begin{marginbox*}{⟨1, 2 or 4 margin values⟩}
  ⟨content⟩
\end{marginbox*}
```

```
margin=⟨all sites⟩
margin=⟨left/right⟩ ⟨top/bottom⟩
margin=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
```

This feature can be used to add a margin (white space) around the content. It can be seen as the opposite of `\trim`. The original baseline of the content is preserved because ⟨*lly*⟩ is added to the depth. It is also available as `marginbox` environment and also usable as `margin` option (see below).

Example:

```
Before \fbox{\marginbox{1ex 2ex 3ex 4ex}{Text}} After
```




```
\marginbox*{⟨all sites⟩}{⟨content⟩}
\marginbox*{⟨left/right⟩ ⟨top/bottom⟩}{⟨content⟩}
\marginbox*{⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩}{⟨content⟩}
```

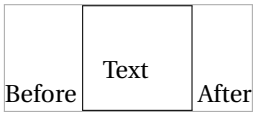
```
\begin{marginbox}{⟨1, 2 or 4 margin values⟩}
  ⟨content⟩
\end{marginbox}
```

```
margin*=⟨all sites⟩
margin*=⟨left/right⟩ ⟨top/bottom⟩
margin*=⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩
```

This starred version is almost identical to the normal `\marginbox`, but also raises the content by the ⟨*lly*⟩ amount, so that the original depth instead of the original baseline is preserved. Note that while `\marginbox` is basically the opposite of `\trim`, `\marginbox*` is not the opposite of `\trim*`. Instead it also takes the same values as the normal value and not view port values like `\trim*`.

Example:

```
Before \fbox{\marginbox*{1ex 2ex 3ex 4ex}{Text}} After
```



3.4 Minimum and Maximum Size

```
\minsizebox{⟨width⟩}{⟨height⟩}{⟨content⟩}
\minsizebox*{⟨width⟩}{⟨totalheight⟩}{⟨content⟩}
```

This macro is like `\resizebox` of the `graphics/x` package, but only resizes the content if its natural size is smaller than the given ⟨*width*⟩ or ⟨*height*⟩. If only one value should be set the other one can be replaced by '`!`'. If required the content is scaled up so that the width and height is equal or larger than the given values, but does not change the aspect ratio. The star variant uses the total height instead of only the height. This macro is used internally for the `min width`, `min height`, `min totalheight` and `min totalsize` options.

Examples:

```
\minsizebox{3cm}{2ex}{Some Text} which will be enlarged
```

Some Text which will be enlarged

```
\minsizebox{!}{4ex}{\fbox{Some Text}} which will be ⁄
    enlarged
```

Some Text which will be enlarged

```
\minsizebox*{!}{4ex}{\fbox{Some Text}} which will be ⁄
    enlarged
```

Some Text which will be enlarged

```
\minsizebox{3cm}{!}{Some Text} which will be enlarged
```

Some Text which will be enlarged

```
\minsizebox{1cm}{1ex}{Some Text}, already large enough
```

Some Text, already large enough

\maxsizebox{⟨*width*⟩}{⟨*height*⟩}{⟨*content*⟩}
\maxsizebox*{⟨*width*⟩}{⟨*totalheight*⟩}{⟨*content*⟩}

This macro is like **\resizebox** of the graphics/x package, but only resizes the content if its natural size is larger than the given ⟨*width*⟩ or ⟨*height*⟩. If only one value should be set the other one can be replaced by '!'. If required the content is scaled down so that the width and height is equal or smaller than the given values, but does not change the aspect ratio. The star variant uses the total height instead of only the height. This macro is used internally for the **max width**, **max height**, **max totalheight** and **max totalsize** options.

Examples:

```
\maxsizebox{1cm}{1ex}{Some Text} which will be reduced
```

Some Text which will be reduced

```
\maxsizebox{!}{1ex}{\fbox{Some Text}} which will be reduced
```

Some Text which will be reduced

```
\maxsizebox*{!}{1ex}{\fbox{Some Text}} which will be ⁄
    reduced
```

Some Text which will be reduced

```
\maxsizebox{1cm}{!}{Some Text} which will be reduced
```

Some Text which will be reduced

```
\maxsizebox{3cm}{1cm}{Some Text}, already small enough
```

Some Text, already small enough

`min width=`⟨*width*⟩
`max width=`⟨*width*⟩
`min height=`⟨*height*⟩
`max height=`⟨*height*⟩
`min totalheight=`⟨*total height*⟩
`max totalheight=`⟨*total height*⟩

These keys allow to set the minimum and maximum width, height or totalheight of the content. The current size of the content is measured and the content is resized if the constraint is not already met, otherwise the content is unchanged. Multiple usages of these keys are checked one after each other, and therefore it is possible that a later one is undoing the size changes of an earlier one. A good example is `max width=\textwidth` which will limit large content to the text width but will not affect smaller content.

`min size={`⟨*width*⟩`}{`⟨*height*⟩`}`
`max size={`⟨*width*⟩`}{`⟨*height*⟩`}`
`min totalsize={`⟨*width*⟩`}{`⟨*total height*⟩`}`
`max totalsize={`⟨*width*⟩`}{`⟨*total height*⟩`}`

These keys allow to specify the minimum or maximum width and (total)height of the content together, which is more efficient than using the width and (total)height keys described earlier.

3.4.1 Scaling

`\scalebox{`⟨*h-factor*⟩`}[`⟨*v-factor*⟩`]{`⟨*content*⟩`}`

With only the mandatory argument the content is evenly scaled accordantly to the given factor. With the optional argument a different vertical scaling factor can be given. This macro is provided by the loaded `graphicx` package and only mentioned here for the sake of completeness. The content is read as normal macro argument and therefore can't hold verbatim or similar special material. An alternative which boxes the content directly is provided by the author's other package `realboxes` as `\Scalebox` and as `Scalebox` environment.

`scale=`⟨*factor*⟩
`scale={`⟨*h-factor*⟩`}{`⟨*v-factor*⟩`}`

The normal `scale` key of `graphicx` only allows for one scale factor which is used for both the horizontal and vertical scaling. With `adjustbox` it is also possible to provide

the horizontal and vertical scale factors separately.

Examples:

```
\adjustbox{scale=2}{Some text!}
```

Some text!

```
\adjustbox{scale={2}{1}}{Some text!}
```

Some text!

`\reflectbox{`⟨*content*⟩`}`

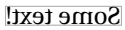
Reflects the content like `\scalebox{-1}[1]` would do. This macro is provided by the loaded `graphicx` package and only mentioned here for the sake of completeness. The content is read as normal macro argument and therefore can't hold verbatim or similar special material. An alternative which boxes the content directly is provided by the author's other package `realboxes` as `\Reflectbox` and as `Reflectbox` environment.

`reflect`

This reflects the content by using `\reflectbox` internally, which is identical to `\scalebox{-1}[1]`, i.e. this key is identical to `scale={-1}{1}`.

Examples:

```
\adjustbox{reflect}{Some text!}
```



3.4.2 Frame

`fbox`
`fbox=`⟨*rule width*⟩
`fbox=`⟨*rule width*⟩ ⟨*sep*⟩
`fbox=`⟨*rule width*⟩ ⟨*sep*⟩ ⟨*margin*⟩

Draws a framed box around the content like `\fbox` would do. Using the optional space separated values the rule width, the separation (inner padding) and the outer margin can be set. If not they default to the values `\fbox` uses by default: `\fboxrule`, `\fboxsep` and zero margin.

Examples:

```
\adjustbox{fbox}{Like \cs{fbox}}
```

Like \fbox

```
\adjustbox{fbox=1pt}{With 1pt rule width}
```

With 1pt rule width

```
\adjustbox{fbox=1pt 2pt}
    {With 1pt rule width and 2pt separation}
```

With 1pt rule width and 2pt separation

```
\adjustbox{fbox={\fboxrule} 1pt}
    {With normal rule width and 1pt separation}
```

With normal rule width and 1pt separation

```
\adjustbox{fbox=1pt 1pt 1pt}
    {With 1pt for rule width, separation and outer margin}
```

With 1pt for rule width, separation and outer margin

`frame`
`frame`=⟨*rule width*⟩
`frame`=⟨*rule width*⟩ ⟨*sep*⟩
`frame`=⟨*rule width*⟩ ⟨*sep*⟩ ⟨*margin*⟩

The `frame` key as the same effect as the `fbox` key but is modeled after LaTeX's `\frame` macro (not the version `beamer` defines). This means it adds a tight frame with zero separation around the content by default. Besides that it accepts the same space separated values. This key is useful to easily add a tight frame around images where the normal separation wouldn't fit.

Examples:

```
\adjustbox{frame}{Tight box}
```

Tight box

`cfbox`=⟨*color*⟩
`cfbox`=⟨*color*⟩ ⟨*rule width*⟩
`cfbox`=⟨*color*⟩ ⟨*rule width*⟩ ⟨*sep*⟩
`cfbox`=⟨*color*⟩ ⟨*rule width*⟩ ⟨*sep*⟩ ⟨*margin*⟩

Identical to `fbox` but uses the given color for the frame. The `xcolor` package must be loaded manually in order for this key to work.

Example:

```
\adjustbox{cfbox=blue 1pt}
    {Like a blue \cs{fbox} with \cs{fboxrule}=1pt}
```

Like a blue \fbox with \fboxrule=1pt

`cframe`=⟨*color*⟩
`cframe`=⟨*color*⟩ ⟨*rule width*⟩
`cframe`=⟨*color*⟩ ⟨*rule width*⟩ ⟨*sep*⟩
`cframe`=⟨*color*⟩ ⟨*rule width*⟩ ⟨*sep*⟩ ⟨*margin*⟩

Identical to `frame` but uses the given color for the frame. The `xcolor` package must be loaded manually in order for this key to work.

Example:

```
\adjustbox{cfbox=blue!50!green}
    {Like a blue and green \cs{frame}}
```

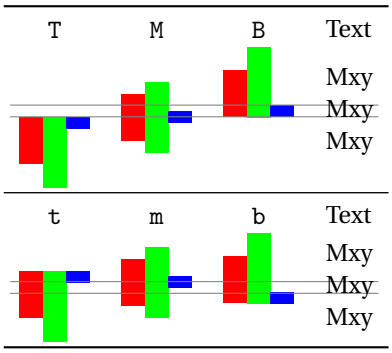
Like a blue and green \frame

3.5 Vertical Alignment

`valign`=⟨*letter*⟩

This key allows to vertically align the content to the top, middle and bottom. The uppercase letters `T`, `M` and `B` align to the content top (i.e. all depth, no height), the geometric, vertical center (equal height and depth) and to the bottom (all height, no depth), respectively. This allows the alignment of content of different size, but will not result in good alignment with text. The lowercase letters `t`, `m` and `b` are aligning the content again to the top, center and bottom but take the current text size in account. The `t` letter leaves a certain height given by the macro[2] `\adjboxvtop` (by default set to the height of `\strut` , i.e. `\ht\strutbox`, which is `.7\baselineskip`), while `b` sets a certain depth given (as negative length) by the macro `\adjboxvbottom` (by default equal to the (negated) `\strut` depth, i.e. `-\dp\strutbox`, which is `.3\baselineskip`). The `m` letter will center towards the vertical center of the text line which is determined by the macro `\adjboxvcenter` (by default `1ex`).

The following table shows the different alignments for three different sized blocks:



[2] A macro and not a length is used to allow for font size relative values like `1ex`.

`raise=`⟨*amount*⟩
`raise={`⟨*amount*⟩`}{`⟨*height*⟩`}`
`raise={`⟨*amount*⟩`}{`⟨*height*⟩`}{`⟨*depth*⟩`}`

This key uses `\raisebox{`⟨*amount*⟩`}{...}` to raise the content upwards for the given ⟨*amount*⟩ (length). A negative length moves the content down. The two optional arguments of `\raisebox{`⟨*amount*⟩`}[`⟨*height*⟩`][`⟨*depth*⟩`]{...}` are also available as optional brace arguments. They can be used to set the official height and depth of the content. This is also possible using the `set height` and `set depth` keys.

Examples:

```
Is \adjustbox{raise=1ex}{higher}
than the normal text
```

Is higher than the normal text

```
Is \adjustbox{raise={1ex}{\height}}{higher}
than the normal text but sill has
its original official height
```

Is higher than the normal text but sill has its original official height

```
Is \adjustbox{raise={1ex}{1ex}{0pt}}{higher and
\rotatebox{-90}{deeper}} but with limited official
height and no depth.
```

Is higher and deeper but with limited official height and no depth.

`set height=`⟨*height*⟩

This sets the official height of the content without actual changing it. This can be seen as a form of trimming. It uses the same internal code as `\raisebox{0pt}[`⟨*height*⟩`]{`⟨*content*⟩`}`.

Example:

```
\adjustbox{set height=.5\height}
    {\shortstack{some stacked\\content}}
```

some stacked content

`set depth=`⟨*depth*⟩

This sets the official depth of the content without actual changing it. This can be seen as a form of trimming. It uses the same internal code as `\raisebox{0pt}[\height][`⟨*depth*⟩`]{`⟨*content*⟩`}`.

Example:

```
\adjustbox{set depth=0pt}
    {\shortstack{some stacked\\content
    with \raisebox{-1ex}{depth}}}
```

some stacked content with depth

`set vsize={⟨height⟩}{⟨depth⟩}`

This sets the official height of depth of the content without actual changing it. This key is simply the combination of `set height` and `set depth`.

Example:

```
\adjustbox{set vsize={2pt}{1pt}}
    {\shortstack{some stacked\\content
    with \raisebox{-1ex}{depth}}}
```

some stacked
content with depth

3.6 Horizontal Alignment

`center`
`center=⟨width⟩`

This key places the content in a horizontal box which is by default `\linewidth` wide (i.e. as wide as a normal text paragraph) and centers it in it. The effect is very similar to `\centerline`. The original content is unchanged, but simply identical white space is added as a left and right margin. This is useful if the content is a figure or table and can be used as a replacement for `\centering`. One important difference is that the content will then have the given width which might influence (sub-)caption placement. If the content is wider than the available width it will stick out on both sides equally without causing an overfull hbox warning. Note that when `\adjustbox` paragraph is used at the beginning of a paragraph the normal paragraph indention is added, which will push the while box to the right and might cause an overfull line. In such cases a `\noindent` must be added beforehand. The `adjustbox` environment already uses this macro.

Examples:

```
\adjustbox{center}{Some content}
```

Some content

```
\adjustbox{center=5cm}{Some content}
```

Some content

`right`
`right=⟨width⟩`

Like `center` this key places the content in a box with the given width (by default `\linewidth`) but right aligns it. If the content is wider than the available width it will stick out into the left side without causing an overfull hbox warning.

Examples:

```
\adjustbox{right}{Some content}
```

Some content

```
\adjustbox{right=5cm}{Some content}
```

Some content

`left`
`left=`⟨*width*⟩

Like `center` this key places the content in a box with the given width (by default `\linewidth`) but left aligns it. If the content is wider than the available width it will stick out into the right side without causing an overfull hbox warning.

Examples:

```
\adjustbox{left}{Some content}
```

Some content

```
\adjustbox{left=5cm}{Some content}
```

Some content

`inner`
`inner=`⟨*width*⟩

Like `center`, `left` and `right` this key places the content in a box with the given width (by default `\linewidth`) but aligns it towards the inner margin. If the content is wider than the available width it will stick into the outer margin without causing an overfull hbox warning. In `twoside` mode this key is equal to `left` for odd pages and equal to `right` for even pages. For `oneside` mode it is always equal to `center`, because there is no inner or outer margin. Note that the page-is-odd test might not always lead to correct results for some material close to a page boundary, because TEX might not have decided on which page it will be placed. This can be improved by loading the `changepage` package with the strict option, which uses a reference to determine the correct page number (and requires the usual additional compiler run).

`outer`
`outer=`⟨*width*⟩

Identical to `inner` but aligns the content towards the outer margin. If the content is wider than the available width it will stick into the outer inner without causing an overfull hbox warning.

3.7 Lapping

The following features can be used to make the content lap over its left or right boundary. This is basically the same as trimming, but provides a different, more dedicated interface.

`\lapbox`[⟨*width*⟩]{⟨*lap amount*⟩}{⟨*content*⟩}

This macro is a generalisation of the LaTeX core macros `\rlap`{⟨*content*⟩} and `\llap`{⟨*content*⟩} which lap the text to the right or left without taking any official space. The `\lapbox` macro can be used to only partially lap the content to the right (positive amount) or left (negative amount). As with all macros of this package the original width can be references using `\width`. The resulting official width of the box is normally the original width minus the absolute lap amount. However, it can also be set explicitly using the option argument. It is also possible to use lap amount which absolute values are larger than the original width. In this case the resulting official width will be zero by default and the content will padded with the required white space. Note that the lap amount always states the distance between the right side of the official box and the right side of the actual content for positive amounts or the distance between the left side of the official box and the left side of the actual content for negative values.

Examples:

General lapping:

```
\lapbox{1cm}{Some Text}
```
Some Text

```
\lapbox{-1cm}{Some Text}
```
Some Text

```
\lapbox[4cm]{1cm}{Some Text}
```
Some Text

```
\lapbox[3cm]{2cm}{Some Text}
```
Some Text

Like `\rlap`:

```
\lapbox[0pt]{\width}{Some Text}
```
Some Text

Like `\llap`:

```
\lapbox[0pt]{-\width}{Some Text}
```
Some Text

A centering `\clap` macro can be achieved using:

```
\lapbox[0pt]{-.5\width}{Some Text}
```
Some Text

```
\lapbox[0pt]{.5\width}{Some Text}
```
Some Text

`lap`=⟨*lap amount*⟩
`lap`={⟨*length*⟩}{⟨*lap amount*⟩}

This wraps the content into a `\lapbox`{⟨*lap amount*⟩}{...} and `\lapbox`[⟨*length*⟩]{⟨*lap amount*⟩}{...}, respectively. Positive ⟨*amounts*⟩ lap the content to the right and negative to the left. The optional ⟨*length*⟩ argument allows to set the final width.

Examples:

```
\adjustbox{lap=.5\width}{Some content}
```
Some content

```
\adjustbox{lap=-.5\width}{Some content}
```
Some content

```
\adjustbox{lap=\width}{Some content}
```
Some content

```
\adjustbox{lap=-\width}{Some content}
```
Some content

```
\adjustbox{lap={\width}{\width}}{Some content}
```
Some content

```
\adjustbox{lap={\width}{-\width}}{Some content}
```
Some content

`rlap`
`llap`

This makes the content to be officially 0pt wide and lap over to the right or left, respectively, like the LaTeX macros `\rlap` and `\llap` do. These are shortcuts for `lap=\width` and `lap=-\width`, respectively. The values for these keys are ignored and should not be used.

Examples:

```
\adjustbox{rlap}{Some content}
```
Some content

```
\adjustbox{llap}{Some content}
```
Some content

3.7.1 Background

`bgcolor=`⟨*color*⟩
`bgcolor={`⟨*model*⟩`}{`⟨*color*⟩`}`

This key adds a colored background to the content. The `xcolor` package (or `color` or `xxcolor`) needs to be loaded as well in order for this to work. The value is passed to an internal `\color` macro.

Examples:

```
\adjustbox{bgcolor=blue}{Text with blue background.}
```
Text with blue background.

```
\adjustbox{bgcolor={rgb}{0 0 1}}{Text with blue background ⟋
    in the RGB color model.}
```
Text with blue background in the RGB color model.

```
\adjustbox{margin=1ex,bgcolor=green}{green with a little ⁄
    more margin}
```

green with a little more margin

```
\adjustbox{margin=1ex,bgcolor=green,margin=1pt,bgcolor=⁄
    yellow}{Emulation of colored frame}
```

Emulation of colored frame

`bgcolor*=`⟨*color macro*⟩

Like `bgcolor` but awaits a full color macro as value. This allows to use other macros as `\color` like `\blendcolors`. See the `xcolor` manual for more details.

Examples:

```
\color{blue}Blue text
\adjustbox{bgcolor*=\blendcolors{!10!yellow}\color{.}}{with⁄
    a yellow-bluish background}
```

Blue text with a yellow-bluish background

```
\color{green}Green text
\adjustbox{bgcolor*=\blendcolors{!10!yellow}\color{.}}{with⁄
    a yellow-greenish background}
```

Green text with a yellow-greenish background

`bgimage=`⟨*image filename*⟩
`bgimage={`⟨*key=value pairs for image*⟩`}{`⟨*image filename*⟩`}`

Adds a background image to the content. The image is stretched if required to fit exactly to the content. It is also possible to provide `\adjustbox` or `\includegraphics` keys to modify the image (before the resizing is done).

`\bgimagebox[`⟨*key=value pairs*⟩`]{`⟨*image filename*⟩`}`

Standalone version of the `bgimage` key. Also available as `bgimagebox` environment.

3.7.2 Pixel size

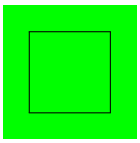
`dpi=`⟨*number (dots per inch)*⟩

The `dpi` key provides a simple interface to set the pixel size to the given DPI (dots per inch) value. For `pdflatex` the length unit `px` can be used to specify pixels. However, the equivalent dimension (length) of one pixel must be set using the `\pdfpxdimen` length register. To set a specific DPI value this length must be set

using `\setlength\pdfpxdimen{1in/`⟨*dots*⟩`}`, which is done by the `dpi=`⟨*dots*⟩ key. Note that the key won't affect the setting for the content but only for the further used keys. However, it is possible to use `\setkeys{adjbox}{dpi=`⟨*number*⟩`}` inside the content or anywhere else in the document to set `\pdfpxdimen` using this interface.

Example:

```
\adjustbox{dpi=72,trim=10px,frame}{%
    \setkeys{adjbox}{dpi=72}%
    \textcolor{green}{\rule{50px}{50px}}%
}
```



`pxdim=`⟨*length*⟩

Alternatively to the `dpi` key the `\pdfpxdimen` length can be set directly to the given value. Afterwards `1px` will stand for the given ⟨*length*⟩.

Example:

```
\adjustbox{pxdim=2pt,trim=2px,frame}
    {\textcolor{green}{\rule{20pt}{20pt}}}
```

3.8 Minipage or other inner environments which change the processing of the content

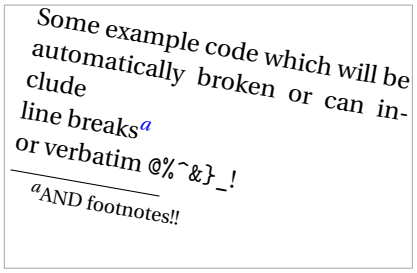
The following keys set the way the content is processed before it is stored it in a box. These keys will overwrite each other and only the latest used key will take effect. Because they affect the inner content directly their order relative to other, normal keys is not meaningful. Also they are only defined for `adjustbox` but do not apply for `\includegraphics`. Because they are therefore only used inside a mandatory argument and never in an optional these keys allow for optional bracket arguments.

`minipage=`⟨*width*⟩
`minipage=[`⟨*position*⟩`][`⟨*height*⟩`][`⟨*inner position*⟩`]{`⟨*width*⟩`}`

This key wraps the inner content in a `minipage` with the given ⟨*width*⟩ before it is stored as horizontal box. Its order relative to other keys is not meaningful (except that future keys of this sub-section will overwrite it). This allows for line breaks and footnotes in the `adjustbox`. All optional arguments of `minipage` are supported. I only the width is given it does not have to be enclosed in braces. The ⟨*position*⟩ argument must be '`t`' for top baseline, '`b`' for bottom baseline and '`c`' for center alignment relative to other text, i.e. defines the resulting baseline. If a ⟨*height*⟩ is defined the ⟨*inner position*⟩ defaults to ⟨*position*⟩ but can also be '`s`' to stretch the content over the whole height. This requires the content to include some vertical stretchable material. Note that all length arguments can include arithmetic expressions like for other keys.

Examples:

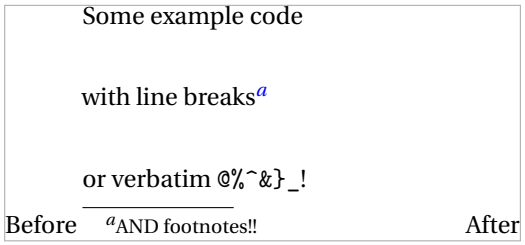
```
\adjustbox{minipage=5cm,angle=-10}{%
   Some example code which will
   be automatically broken or can include \\
   line breaks\footnote{AND footnotes!!}\\
   or verbatim \verb+@%^&}_+!%
}
```




```
Before \begin{adjustbox}{minipage=[b][3cm][s]{5cm}}
   Some example code

   \vfill
   with line breaks\footnote{AND footnotes!!}

   \vfill
   or verbatim \verb+@%^&}_+!%
\end{adjustbox} After
```



`tabular`=[⟨*position*⟩]{⟨*column specification*⟩}
`tabular*`=[⟨*position*⟩]{⟨*width*⟩}{⟨*column specification*⟩}
`array`=[⟨*position*⟩]{⟨*column specification*⟩}

Places the content in a `tabular`, `tabular*` or `array` environment, respectively. These keys require different implementations for macro (`\adjustbox`) and environment mode (`adjustbox` environment) in order to insert the end code correctly. Note that the environment mode is more efficient and fully stable, while the macro mode requires the last row to end with an explicit \\ (which can be followed by `\hline` or any other macro which uses `\noalign` internally). In macro mode the \\ is internally redefined to check for the closing brace. While this was successful tested for normal usages it might still cause issues with unusual or complicated cases. Note that these environments are taken as part of the content and so the usage of arithmetic expressions for length arguments is not supported.

Examples:

```
\adjustbox{tabular=lll}{%
                  \hline
    A & B & C \\\hline
    a & b & c \\\hline
}
```

A	B	C
a	b	c

```
\begin{adjustbox}{tabular=lll}
    A & B & C \\
    a & b & c
\end{adjustbox}
```

A	B	C
a	b	c

`stack`
`stack=`⟨*horizontal alignment*⟩
`stack={`⟨*horizontal alignment*⟩`}{`⟨*vertical alignment*⟩`}`

`\stackbox[`⟨*horizontal alignment*⟩`][`⟨*vertical alignment*⟩`]{`⟨*content*⟩`}`

```
\begin{stackbox}[⟨horizontal alignment⟩][⟨vertical alignment⟩]
  ⟨content⟩
\end{stackbox}
```

The `stack` key and its corresponding macro and environment can be used to stack multiple lines similar to the `\shortstack` macro, but both the horizontal and vertical alignment can be selected by a single letter each. Also a proper baseline skip is inserted. This is implemented using the `varwidth` environment which is based on the `minipage` environment. Its maximal width arguments is fixed internally to `\linewidth`.

Possible horizontal alignments are: '`l`' (left), '`r`' (right), '`c`' (centered, default), '`j`' (justified). Possible vertical alignments are the same as for `minipage`: '`t`' (top baseline), '`b`' (bottom baseline, default), '`c`' (vertical centered). Because these arguments are always single letters the '`{ }`' around them can be skipped, so that the value can simple be two concatenated letters.

Example:

```
.\adjustbox{stack}{A\\B\\CC}.
```

.A
B
CC.

```
.\adjustbox{stack=r}{A\\B\\CC}.
```

.A
B
CC.

```
.\adjustbox{stack=ct}{A\\B\\CC}.
```

. A .
B
CC

`innerenv`=⟨*environment name*⟩
`innerenv`={⟨*environment name*⟩}⟨*environment options*⟩

Wraps the inner content in the given ⟨*environment*⟩ before it is stored as horizontal box. It should be kept in mind that there is some internal code between the begin of the environment and the content. For this reason a `tabular`, `array` or similar environment will not work here, because that code will be taken as part of the first cell. Note that such a environment is taken as part of the content and so the usage of arithmetic expressions for length arguments is not supported.

Example:

```
\newenvironment{myenv}[2][]{Before [#1](#2)}{After}
\adjustbox{innerenv={myenv}[ex]{amble}}{Content}
```

Before [ex](amble)ContentAfter

```
\adjustbox{innerenv={myenv}{amble}}{Content}
```

Before [](amble)ContentAfter

`innercode`={⟨*begin code*⟩}{⟨*end code*⟩}

Places the given code before and after the inner content before it is stored as horizontal box. Note that such code is taken as part of the content and so the usage of arithmetic expressions for length arguments is not supported.

Example:

```
\adjustbox{innercode={\color{green}}{!}}{Content}
```

Content!

3.9 Adding own Code or Environments

`env`=⟨*environment name*⟩
`env`={⟨*environment name*⟩}⟨*environment options*⟩

Adds an ⟨*environment*⟩ around the content and the already existing code around it which was added by other keys beforehand. Potential ⟨*environment options*⟩ (or any other code) can follow the environment name if it was set inside braces. At this stage the content is already boxed and format macros won't have any effect on any included text. For this the `innerenv` key needs to be used instead.

`addcode`={⟨*code before*⟩}{⟨*code afters*⟩}

Adds some ⟨*code before*⟩ and some ⟨*code after*⟩ the content and the already existing code around it which was added by other keys beforehand. At this stage the content is already boxed and format macros won't have any effect on any included text.

`appcode=`⟨*code afterwards*⟩

Appends come ⟨*code after*⟩ the content and the already existing code around it which was added by other keys beforehand. More complex code should be enclosed in braces.

`precode=`⟨*code before*⟩

Prepends come ⟨*code afterwards*⟩ the content and the already existing code around it which was added by other keys beforehand. More complex code should be enclosed in braces.

`execute=`⟨*code*⟩

Simply executes the code immediately. This is done in the key processing phase and is intended mostly for debugging purposes. Previous (normal) keys won't have an effect yet.

`Execute=`⟨*code*⟩

Simply executes the code immediately. This is done in the key processing phase for inner environments (see subsection 3.8) and is intended mostly for debugging purposes. Only previously used special keys for modifying the boxing of the content will have an effect yet. All other keys are not yet processed.

4 Other

`\phantombox{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*depth*⟩`}`

This macro produces an empty box with the given width, height and depth. It is equivalent to `\phantom{\rule[-`⟨*depth*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`+`⟨*depth*⟩`}}` but more efficient and more user friendly.

Example:

```
Before \fbox{\phantombox{1cm}{2ex}{1ex}} After
```

Before [ ] After