The adjustbox Package

Martin Scharrer

martin.scharrer@web.de

Version v1.3c – 2025/02/26

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

VC: https://github.com/MartinScharrer/adjustbox/

Abstract

This package allows to adjust general (LA)TEX material in several ways using a key=value interface. It got inspired by the interface of `\includegraphics` from the `graphicx` package. This package also loads the `trimclip` package which code was once included in this package.

Contents

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. The `\clipbox` and `\trimbox` macros were then moved together with the required output driver code to a dedicated package `trimclip`. This allows documents and other packages to use these basic features without loading the full `adjustbox` package.

1.1 Dependencies

The `adjustbox` package and its `trimclip` sub-package require the author's other packages `adjcalc` (bundled with `adjustbox`) and `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` and the `stackbox` environment. The `ifoddpage` package is automatically loaded if installed, otherwise the `outer` and `inner` keys are disables as well as `\outersidebox` and `\innersidebox` macros and corresponding environments. 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 to be loaded manually.

1.2 Verbatim Support

The macros provided by `adjustbox` and `trimclip` read the content as a horizontal TEX box and not as an macro argument in order to support verbatim content. This means that the braces around the content can also be written as `\bgroup` and `\egroup`:

> `\adjustbox`{key=value,...}`\bgroup` ⟨*content*⟩`\egroup`

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

> `\adjustbox`{key=value,...}`\somemacro`

This is to support the habit of some LaTeX users to drop the braces for single token arguments. All environments support verbatim content. Note that the plainTEX syntax for environments (`\adjustbox` . . . `\endadjustbox`) can not be used because it will trigger `\adjustbox` as a macro. However, the braces around the content can be replaced by `\bgroup` and `\egroup`.

2 Package Options

This package accepts the following options:

export Exports the keys of `\adjustbox` to `\includegraphics` so that they can be used with this macro as well. Note that not all keys will works properly with `\includegraphics` as its internal code does not support all features. If problems occur the option Export should be used. For new documents it is recommended to use the macro `\adjustimage`{⟨*keys*⟩}{⟨*filename*⟩} instead.

Export Sets `\includegraphics` to be identical to `\adjincludegraphics`, which also allows the usage of all `\adjustbox` keys.

pgf This option is passed to `trimclip` and makes it to use the `pgf` package for all clip operations. This overrides all automatically detected drivers.

PGF This option will pass the pgf to `trimclip` and enable the pgfmath option of adjcalc, i.e. the `pgf` package is used for clip operations and the `pgfmath` pacakge is used to parse length arguments.

minimal Only load a minimal set of code, i.e. `trimclip` and `\adjustbox`/adjustbox but no additional keys and macros.

The following options define the way length values are processed by the provided macros. They are passed to the `\adjcalcset` macro of the adjcalc package and load any required packages. These options can also be used as local keys on `\adjustbox`. See the adjcalc manual for more details on them.

etex Uses the ϵ-TeX primitive `\glueexpr` to parse length values.

calc Uses the calc package to parse length values.

pgfmath Uses the pgfmath package of the pgf bundle to parse length values.

defaultunit=⟨*unit*⟩ This sets the default unit used for values which accept length without units, e.g. `trim`.

In addition, all driver options of the `trimclip` package are also accepted and passed to it. All unknown options are passed to the `graphicx` package, which might pass them down to the `graphics` package.

3   Main Macros

The following macros and environments represent the main functionality of this package. 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.

3.1   Adjust Box Content

```
\adjustbox{⟨key=value, …⟩}{⟨content⟩}
```

The `\adjustbox` macro allows the modification of general (LA)TEX content using a key=value syntax. This can be seen as an extension of the `\includegraphics` macro from images to any (LA)TEX material. The macro supports the all `\includegraphics` keys and many more. However, they are provided as a mandatory not as an optional argument, because an `\adjustbox` without options would not make sense and can be replaced by a simple `\mbox`. As already stated the content is read as a box, not as a macro argument, therefore verbatim or other special content is supported and the braces can also be replaced by `\bgroup` and `\egroup`. There is no starred version of this macro.

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

The environment version of `\adjustbox`. A difference to the macro is that it includes `\noindent` to suppress paragraph indention.

3.2 Adjust Images

> **\adjustimage**{⟨*key=value, . . .*⟩}{⟨*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*⟩}}, which is the definition of **\adjustimage**. However, it does not use **\includegraphics** directly, but an internal macro, to allow the redefinition of **\includegraphics**. This macro 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**, i.e. with an optional argument. This macro allows to use all features of **\adjustbox** on images, however special care must be taken if the values include '[ ]'. In this case either the whole value or the while optional argument must be wrapped in '{ }':
\adjincludegraphics[key={[..]},...]{..} or
\adjincludegraphics[{key=[..],...}]{..}.
It is possible to redefine **\includegraphics** to **\adjincludegraphics** and this is done by the Export option (not to be confused with the export option).

---

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

3.3 Defining custom environments and macros

After `adjustbox` is used to define custom environments to format specific content. If an environment contains other environments these are often written in the plain form `\foo` ... `\endfoo` to avoid the overhead of `\begin` and `\end` and to keep the outer environment name for meaningful error messages. As mentioned, the dual nature of `\adjustbox` which will detect if it is used as environment of macro does not permit this plain form, which can be a pitfall for custom environment definitions. Instead the macro form `\adjustbox`{⟨*key=value*⟩}`\bgroup` ... `\egroup` should be used. To simplify custom environment definitions based on `\adjustbox`/adjustbox the following macros are provided.

`\newadjustboxenv`{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

This declaration defines an environment and macro with the given name which will apply the given adjustbox keys to its content. The environment/macro can also posses arguments, including one leading optional argument, which can be used inside the key=value pairs. It is recommended to wrap the arguments into braces, e.g. use '{#1}' instead of '#1' etc., to avoid issues with potentially included commas and equal signs. The number and type of arguments are defined using two optional arguments in the same way as for `\newcommand` and `\newenvironment`.

`\newadjustboxenv`∗{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

This declaration works similar to the unstarred version but only defines an environment. This environment can therefore be used like any other normal LaTeX environment, including in its plain form `\`⟨*name*⟩ ... `\end`⟨*name*⟩.

`\renewadjustboxenv`{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
`\renewadjustboxenv`∗{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like `\newadjustboxenv` and `\newadjustboxenv`∗ but will redefine an existing macro/environment and cause an error if it was not yet defined.

`\provideadjustboxenv`{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
`\provideadjustboxenv`∗{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like `\newadjustboxenv` and `\newadjustboxenv`∗ but will define a macro/environment only if it does not exists yet.

`\declareadjustboxenv`{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
`\declareadjustboxenv`∗{⟨*name*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like `\newadjustboxenv` and `\newadjustboxenv`∗ but will always define a macro/environment even if it does already exist.

\newadjustboxcmd{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

This declaration defines a macro which applies the given `adjustbox` keys to its content. As with **\adjustbox** the content is read as box, not as macro argument. The macro arguments (if any) are defined in the same way as with **\newcommand**. The arguments can then be used inside the key=value pairs. The number of arguments does not include the content. This declaration is intended if no environment form is required and is more efficient than the dual declaration.

Example:

```
\newadjustboxcmd{\frotate}[1]{rotate={#1},fbox}%
\frotate{30}{Text}
```

\renewadjustboxcmd{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like **\newadjustboxcmd** but will redefine an existing macro and cause an error if it was not yet defined.

\provideadjustboxcmd{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like **\newadjustboxcmd** but will define the macro only if it does not exist yet.

\declareadjustboxcmd{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

Like **\newadjustboxcmd** but will define the macro in any case even if it does exist yet.

\newadjustimage{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
\renewadjustimage{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
\provideadjustimage{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}

These macros allow to define new versions of **\adjustimage** with predefined key lists. The new define image macros can also have arguments which can be used in the predefined key list to substitude values or one or multiple keys.

They work like **\newcommand**, **\renewcommand** and **\providecommand**, respectively, where ⟨*num*⟩ is the number of arguments and ⟨*default*⟩ provides a default value for the then optional first argument. However, instead of a macro content a ⟨*key=value*⟩ list must be present, which can contain the arguments (#1, #2, etc). If arguments are used as values they should be wrapped in braces. If there are only part of a value the whole value should be wrapped in braces. This is to avoid causes commas inside the arguments to cause issues with the key=value list.

Example:

```
\newadjustimage{\myimage}[2][red]{width={#2},cfbox={#1}}%
\myimage{2cm}{example-image}
\myimage[blue]{2cm}{example-image}
```
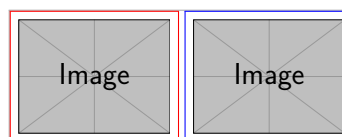

\declareadjustimage{⟨*\macro*⟩}[⟨*num*⟩][⟨*default*⟩]{⟨*key=value,…*⟩}
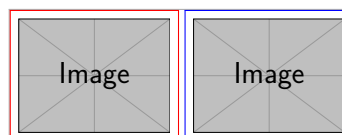
Like **\newadjustimage** but will not cause an error if the macro is already defined.

\NewAdjustImage{⟨*\macro*⟩}{⟨*xparse argument specification*⟩}{⟨*key=value,…*⟩}
\RenewAdjustImage{⟨*\macro*⟩}{⟨*xparse argument specification*⟩}{⟨*key=value,…*⟩}
\ProvideAdjustImage{⟨*\macro*⟩}{⟨*xparse argument specification*⟩}{⟨*key=value,…*⟩}
\DeclareAdjustImage{⟨*\macro*⟩}{⟨*xparse argument specification*⟩}{⟨*key=value,…*⟩}

These macros also allow to define new versions of **\adjustimage** with predefined key lists, but use the xparse package (which must be loaded separately!) and its macros **\NewDocumentCommand**, **\RenewDocumentCommand**, **\ProvideDocumentCommand** and **\DeclareDocumentCommand** instead of the standard LaTeX macro creation macros. This allows a larger variety of optional and mandatory arguments. Please see the xparse manual for more details.

Example:

```
\NewAdjustImage{\myimage}{O{red}m}{width={#2},cfbox={#1}}%
\myimage{2cm}{example-image}
\myimage[blue]{2cm}{example-image}
```

3.4 Setting keys globally

```
\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}
```

3.5 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.

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 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.

`\smallestside`

This macro expands to either `\width` or `\totalheight` whatever is smaller.

`\largestside`

This macro expands to either `\width` or `\totalheight` whatever is larger.

`\Smallestside`

This macro expands to either `\Width` or `\Totalheight` whatever is smaller.

`\Largestside`

This macro expands to either `\Width` or `\Totalheight` whatever is larger.

Default unit

If no unit is provided for of the bounding box coordinates (left, bottom, right, top) 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).

4 Adjustbox Keys

This packages provides the following `\adjustbox` keys with matching macros and environments.

4.1 Trimming and Clipping

The following keys allow content to be trimmed (i.e. the official size is made smaller, so the remaining material laps over the official boundaries) or clipped (overlapping material is not displayed). These keys come in different variants, where the lower-case keys represent the behavior of the corresponding `\includegraphics` keys. The corresponding macros (`\trimbox`, `\clipbox`, etc.) and environments (`trimbox`, `clipbox`, etc.) are included in the accompanying `trimclip` package and are explained in its manual.

`trim`=⟨*all sites*⟩
`trim`=⟨*left/right*⟩ ⟨*top/bottom*⟩
`trim`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩

This key represents the original `trim` key of `\includegraphics` but accepts its value in different forms. Unlike most other keys it always acts on the original content independent in which order it is used with other keys. If only one value is given it will be used for all four sites. If only two values are given, seperated by a space, the first one will be used for the left and right side and the second for the bottom and top side.

`viewport`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩

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

`clip`
`clip`=true|false

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. Note that the material will still be part of the output file but is simply not shown. It might be exported from the output file using special tools, so using it to censor classified information is a bad idea.

`Trim`=⟨*all sites*⟩
`Trim`=⟨*left/right*⟩ ⟨*top/bottom*⟩
`Trim`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩
`Viewport`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩

The normal `trim` and `viewport` keys are applied on the original content before any resizing or (most) other keys take effect. 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 only 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.

`Clip`=⟨*all sites*⟩
`Clip`=⟨*left/right*⟩ ⟨*top/bottom*⟩
`Clip`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩
`Clip*`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩

The `Clip` key will clip the given amounts from the current content and can be used several times if required. The starred version will use the given coordinates as `viewport`. These keys work by wrapping the content internally in a `\clipbox` or `\clipbox`∗ macro.

`rndcorners`=⟨*radius for all four corners*⟩
`rndcorners`=⟨*left corners*⟩ ⟨*right corners*⟩
`rndcorners`=⟨*upper left*⟩ ⟨*upper right*⟩ ⟨*lower left*⟩ ⟨*lower right*⟩

This key produces round corners by clipping the content. This is done using the same way as the above clipping keys but instead of a rectangle clipping path one with quarter circles is used. The radius of the every round corner can be given separately if wanted. If only one value is given it is used for all four corners. If only two values are given they will be used for the left and right side, respectively. A value of 0pt will produce a normal rectangular corner. No radius should be larger than half the smallest side (minimum of `\width` and `\totalheight`).

Please see also the related keys `rndfbox` and `rndframe`, which allow to add a rounded frame around the content as well.
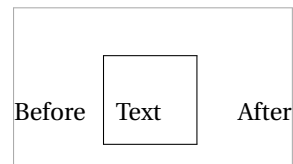
4.2 Margins and Vertical Spacing

```
margin=⟨all sites⟩
margin=⟨left/right⟩ ⟨top/bottom⟩
margin=⟨left⟩ ⟨bottom⟩ ⟨right⟩ ⟨top⟩
```

This key 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 ⟨*bottom*⟩ is added to the depth. It is also available under the alternative name `padding`, which can be used to more visually distinguish an inner margin from an outer margin e.g. if a frame is added.

Example:

```
Before \adjustbox{padding=1ex 2ex 3ex 4ex,frame,margin=1ex 2ex 3ex 4ex}{Text} After
```



```
margin*=⟨all sites⟩
margin*=⟨left/right⟩ ⟨top/bottom⟩
margin*=⟨left⟩ ⟨bottom⟩ ⟨right⟩ ⟨top⟩
```

This starred version is almost identical to the normal `margin` key, but also raises the content by the ⟨*bottom*⟩ amount, so that the original depth is preserved instead of the original baseline. Note that while `margin` is basically the opposite of `Trim`, `margin*` is not the opposite of `Trim*`. Instead it also takes the same values as the unstarred key and not viewport values like `Trim*`. An alternative name of `margin*` is `padding*`.

Example:

```
Before \adjustbox{padding*=1ex 2ex 3ex 4ex,frame,margin*=1ex 2ex 3ex 4ex}{Text} Afte
```



```
vspace=⟨above/below⟩
vspace=⟨above⟩ ⟨below⟩
```
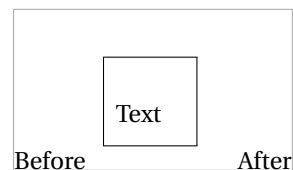
The `vspace` key adds a vertical space before and after the content. This is done by adding paragraph breaks and `\vspace` macros, which will yield better spacing between paragraphs then when using the `margin` key. However `vspace` forces vertical mode around the content. Adding further keys after `vspace` will force restricted

horizontal mode again and the vertical space will be ignored. For this situation the `margin` key is better suited.

---

vspace*=⟨*above/below*⟩
vspace*=⟨*above*⟩ ⟨*below*⟩

Identical to **vspace** but uses **\vspace*** instead. The difference is that the vertical space is not ignored at page breaks, but its full amount is always forced.

4.3 Minimum and Maximum Size

The following keys allow to set a minimum or maximum size. The content will be scaled down or up if required.

```
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.

```
\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` keys.

Examples:

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

$$\boxed{\text{Some Text} \text{which will be enlarged}}$$

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

$$\boxed{\boxed{\text{Some Text}} \text{which will be enlarged}}$$

```
\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` keys.

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

---

warn width
warn width=⟨*max width*⟩

If this key is used the current width of the content is measured and compared with the given maximum width (default is **\linewidth**). If the content is wider than this value by more than the TeX length **\hfuzz** (by default 0.1pt) an *Overfull hbox* warning

is produced. The warning will include the source code position and the information that it was an `adjustbox` not a normal paragraph:

```
Overfull \hbox (X.Ypt too wide) in adjustbox at line N
Overfull \hbox (X.Ypt too wide) in adjustbox at lines N-M
```

Note that this key is not able to take any horizontal space before the `adjustbox` into account. This includes a paragraph indention which might be present with `\adjustbox`.

This key is useful to be used before the horizontal alignment keys which change the official width of the content and prevent the normal *Overfull* warnings to take affect.

4.4 Scaling

---

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!}
```
$$\text{Some text!}$$

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

---

reflect

This key 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!}
```
Some text!

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

These macros is provided by the loaded graphicx package and only mentioned here for the sake of completeness. See the grfguide for more details.

4.5 Frame

The following keys can be used to draw frames around the content.

```
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 has 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{cframe=blue!50!green}
    {Like a blue and green \cs{frame}}
```

Like a blue and green \frame

```
rndframe=⟨radius for all four corners⟩
rndframe=⟨left corners⟩ ⟨right corners⟩
rndframe=⟨upper left⟩ ⟨upper right⟩ ⟨lower left⟩ ⟨lower right⟩
rndframe={⟨options⟩}{⟨radius for all four corners⟩}
rndframe={⟨options⟩}{⟨left corners⟩ ⟨right corners⟩}
rndframe={⟨options⟩}{⟨upper left⟩ ⟨upper right⟩ ⟨lower left⟩ ⟨lower right⟩}
```

```
color={⟨color⟩}
color*={⟨color command⟩}
width={⟨rule width⟩}
sep={⟨rule separation⟩}
```

```
clip
clip=true|false
```

Example:

```
\adjustbox{scale=3,bgcolor=gray,rndframe={color=red,width=2\fboxrule,sep=2\fboxsep}
    {\Huge XXXX}%
```



```
rndcornersbox
```

Like **rndframe**(see argument list there) but only clips the corners without
drawing a frame.

```
rndfbox
```

Like **rndframe**(see argument list there) but with preset **sep**= 0pt.

```
\rndcornersbox
\rndframebox
\rndfbox
```

Standalone macro versions of the corresponding keys. See there for the list of argu-
ments.

4.6 Vertical Alignment

The following keys can be used to adjust the vertical alignment and size of the content.

---
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:

---
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

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

```
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

4.7 Horizontal Alignment

The following keys can be used to adjust the horizontal alignment and size of the content.

---
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

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

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 equal to `left`. 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).

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

This key centers the content relative to the page and independent from the text area. Afterwards the content has the same distance to the left and right page borders. It differs from the `center` key when the left and right page margins are different which is the case in twoside documents or in environments which change the margins.

Note that the content will look misalign to the text paragraphs before and afterwards and is therefore only pleasing for overwide images or similar content.

The optional width defaults to `\linewidth` and determines the official width of the content. If a smaller value is selected following material will be placed on

the same line while maybe overlapping the content. Larger values will cause an `Overfull \hbox` warning.

```
pageleftalign
pageleftalign=⟨width⟩
```

Similar to `pagecenter` (see remarks there) but aligns the content on the left side of the page instead of centering it.

```
pagerightalign
pagerightalign=⟨width⟩
```

Similar to `pagecenter` (see remarks there) but aligns the content on the left side of the page instead of centering it.

```
pageinner
pageinner=⟨width⟩
```

Similar to `pagecenter` (see remarks there) but aligns the content on the inner side of the page instead of centering it. This assumes that the document is compiled in `twoside` mode. In `oneside` mode the macro is identical to `pageleftalign`, i.e. always aligns to the left of the page.

```
pageouter
pageouter=⟨width⟩
```

Similar to `pagecenter` (see remarks there) but aligns the content on the outer side of the page instead of centering it. This assumes that the document is compiled in `twoside` mode. In `oneside` mode the macro is identical to `pagerightalign`, i.e. always aligns to the right of the page.

```
textareacenter
textareacenter=⟨width⟩
```

This key centers the content on the text area (the box with size `\textwidth` × `\textheight` containing the text on a page) even if the box is inside an indending environment like `enumerate`. For this it is assumed that the adjusted box is on the start of a line or paragraph. If this is not the case the content will be overlapping the material on its left side.

The optional width defaults to `\linewidth` and determines the official width of the content. If a smaller value is selected following material will be placed on the same line while maybe overlapping the content. Larger values will cause an `Overfull \hbox` warning.

```
textareatalign
textareatalign=⟨width⟩
```

Similar to `textareacenter` (see remarks there) but aligns the content on the left side of the text area instead of centering it.

```
textareatalign
textareatalign=⟨width⟩
```

Similar to `textareacenter` (see remarks there) but aligns the content on the left side of the text area instead of centering it.

```
textareainner
textareainner=⟨width⟩
```

Similar to `textareacenter` (see remarks there) but aligns the content on the inner side of the text area instead of centering it. This assumes that the document is compiled in `twoside` mode. In `oneside` mode the macro is identical to `pageleftalign`, i.e. always aligns to the left of the text area.

```
textareaouter
textareaouter=⟨width⟩
```

Similar to `textareacenter` (see remarks there) but aligns the content on the outer side of the text area instead of centering it. This assumes that the document is compiled in `twoside` mode. In `oneside` mode the macro is identical to `pagerightalign`, i.e. always aligns to the right of the text area.

```
noindent
```

This key will issue an `\noindent` for the `\adjustbox` to avoid a paragraph indention if it starts a paragraph. It does not expect a value and will ignore it if provided.

```
leavevmode
```

This key will issue an `\leavevmode` for the `\adjustbox` to enter paragraph mode, i.e. it starts a paragraph. This is usually done automatically by all Keys, but is provided as a failsafe in case of any bugs.

```
\centerbox[⟨width⟩]{⟨content⟩}
```

```
\begin{centerbox}[⟨width⟩]
    ⟨content⟩
\end{centerbox}
```

Macro and environment version of the `center` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\leftalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{leftalignbox}[⟨width⟩]
   ⟨content⟩
\end{leftalignbox}
```

Macro and environment version of the **left** key. The optional width argument defaults to **\linewidth**. Both include a **\noindent** to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\rightalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{rightalignbox}[⟨width⟩]
   ⟨content⟩
\end{rightalignbox}
```

Macro and environment version of the **rightalign** key. The optional width argument defaults to **\linewidth**. Both include a **\noindent** to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\innersidebox[⟨width⟩]{⟨content⟩}
```

```
\begin{innersidebox}[⟨width⟩]
   ⟨content⟩
\end{innersidebox}
```

Macro and environment version of the **inner** key. The optional width argument defaults to **\linewidth**. Both include a **\noindent** to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\outersidebox[⟨width⟩]{⟨content⟩}
```

```
\begin{outersidebox}[⟨width⟩]
   ⟨content⟩
\end{outersidebox}
```

Macro and environment version of the **outer** key. The optional width argument defaults to **\linewidth**. Both include a **\noindent** to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\centerpagebox[⟨width⟩]{⟨content⟩}
```

```
\begin{centerpagebox}[⟨width⟩]
   ⟨content⟩
\end{centerpagebox}
```

Macro and environment version of the `centerpage` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\pagecenterbox[⟨width⟩]{⟨content⟩}
```

```
\begin{pagecenterbox}[⟨width⟩]
   ⟨content⟩
\end{pagecenterbox}
```

Macro and environment version of the `pagecenterbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\pageleftalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{pageleftalignbox}[⟨width⟩]
   ⟨content⟩
\end{pageleftalignbox}
```

Macro and environment version of the `pageleftalignbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\pagerightalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{pagerightalignbox}[⟨width⟩]
   ⟨content⟩
\end{pagerightalignbox}
```

Macro and environment version of the `pagerightalignbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\pageinnerbox[⟨width⟩]{⟨content⟩}
```

```
\begin{pageinnerbox}[⟨width⟩]
  ⟨content⟩
\end{pageinnerbox}
```

Macro and environment version of the `pageinnerbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\pageouterbox[⟨width⟩]{⟨content⟩}
```

```
\begin{pageouterbox}[⟨width⟩]
  ⟨content⟩
\end{pageouterbox}
```

Macro and environment version of the `pageouterbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\textareacenterbox[⟨width⟩]{⟨content⟩}
```

```
\begin{textareacenterbox}[⟨width⟩]
  ⟨content⟩
\end{textareacenterbox}
```

Macro and environment version of the `textareacenterbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\textareatalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{textareatalignbox}[⟨width⟩]
  ⟨content⟩
\end{textareatalignbox}
```

Macro and environment version of the `textareatalignbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\textareatalignbox[⟨width⟩]{⟨content⟩}
```

```
\begin{textareatalignbox}[⟨width⟩]
   ⟨content⟩
\end{textareatalignbox}
```

Macro and environment version of the `textareatalignbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\textareainnerbox[⟨width⟩]{⟨content⟩}
```

```
\begin{textareainnerbox}[⟨width⟩]
   ⟨content⟩
\end{textareainnerbox}
```

Macro and environment version of the `textareainnerbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

```
\textareaouterbox[⟨width⟩]{⟨content⟩}
```

```
\begin{textareaouterbox}[⟨width⟩]
   ⟨content⟩
\end{textareaouterbox}
```

Macro and environment version of the `textareaouterbox` key. The optional width argument defaults to `\linewidth`. Both include a `\noindent` to avoid paragraph indention. If an paragraph indention is wanted it can be created by placing a \mbox{} or the identical \null macro before it.

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.

---

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

This key will make the content lap over the surrounding elements to the right or left by the given amount.. A positive amount will make the content lap over to the right and a negative one to the left. The optional ⟨*length*⟩ argument allows to set the final width. If not used the resulting width will be the original width minus the absolute lap amount. This key internally uses **\lapbox**{⟨*lap amount*⟩}{...} and **\lapbox**[⟨*length*⟩]{⟨*lap amount*⟩}{...}, respectively.

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
clap

This makes the content to be officially 0pt wide and lap over to the right or left, respectively, like the LATEX macros **\rlap**, **\llap** and **\clap** do. These are shortcuts for **lap**=**\width**, **lap**=-**\width** and **lap**={0pt}{-0.5**\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

`\adjustbox{clap}{Some content}`   Some content

`\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

4.8   Support for `graphicx` keys

```
bbllx=⟨length⟩
bblly=⟨length⟩
bburx=⟨length⟩
bbury=⟨length⟩
decodearray
intent
interpolate
maskarray
natheight=⟨length⟩
natwidth=⟨length⟩
ocobjnum
ocobjref
page=⟨page numbers⟩
pagebox
quiet
resetcolor
resolution
xmpfile
```

The above `graphicx` keys are also supported by `\adjustbox`. This is done by relying on the original `graphicx` code, i.e. the key code is copied into the `adjustbox` namespace. These keys are therefore mostly for use with images and might not work well with other content. For more details on these keys see the `graphicx` manual.

4.9 Colors

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

This key sets the color for all material. Both the content and further material added by other keys will be affected. The color change is immediate and will occur before basically every other key takes effect. The order in which this key is used is therefore meaningless. Using this key several times will cause several color changes with the last one be used for the remaining material. Note that `color` will not influence any `fgcolor` or `bgcolor` beside the fact that it changes the *current color* accessible using the special name '.'.

```
color*=⟨color macro⟩
```

Like `color` 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. A `\blendcolors` will affect both `fgcolor` and `bgcolor`.

```
fgcolor=⟨color⟩
fgcolor={⟨model⟩}{⟨color⟩}
```

This key sets the color for the content. The value is passed to an internal `\color` macro. This is done using `innercode` because the color must be set before the content is boxed. Further material added using keys, e.g. a `frame`, is not affected by this key. The xcolor package (or color or xxcolor) needs to be loaded otherwise an error will be raised.

```
fgcolor*=⟨color macro⟩
```

Like `fgcolor` 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.

```
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}
```

<div style="text-align:center">green with a little more margin</div>

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

<div style="text-align:center">Emulation of colored frame</div>

---

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 backgrou
```

<div style="text-align:center">Blue text with a yellow-bluish background</div>

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

<div style="text-align:center">Green text with a yellow-greenish background</div>

\bgcolorbox[⟨*model*⟩]{⟨*color*⟩}
\bgcolorbox∗{⟨*color macro*⟩}

```
\begin{bgcolorbox}[⟨model⟩]{⟨color⟩}
   ⟨content⟩
\end{bgcolorbox}
```

```
\begin{bgcolorbox*}{⟨color macro⟩}
   ⟨content⟩
\end{bgcolorbox*}
```

Standalone macro and environment versions of the `bgcolorbox` and `bgcolorbox*` keys.

4.10 Background and Forground Images and Text

The following keys, macros and environment allow to add background and foreground layers. These layers are placed on the baseline of the original content and are horizontally centered, but can be adjusted using own `\adjustbox` or `\includegraphics` keys. The following dimension registers can be used to refer to the dimensions of the original content.

`\pwidth` `\pheight` `\pdepth` `\ptotalheight`

These dimension registers hold the width, height, depth and total height (height+depth) of the parent box and can be used to resize a background or foreground layer accordantly. They are only defined inside the key=value arguments of the following keys, macros and environments.

`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).

Examples:

```
\adjustbox{bgimage=example-grid-100x100bp}{Some text}
```

Some text

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

Adds a background image to the content like `bgimage` but the image is not stretched to fit the content. Instead it is added in its natural size but can be adjusted using `\adjustbox` or `\includegraphics` keys. The image is placed on the baseline of the content and horizontally centered. It can be lowered to the lower edge of the content using `raise`=-`\pdepth` and left aligned using `left`=`\pwidth` etc.

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

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

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

Standalone macro and environment versions of the `bgimage` and `bgimage*` keys.

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

Adds a foreground image to the content. The image is stretched 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). The image is assumed to have transparent elements, otherwise the existing content is fully covered.

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

Adds a foreground image to the content like `fgimage` but the image is not stretched to fit the content. Instead it is added in its natural size but can be adjusted using `\adjustbox` or `\includegraphics` keys. The image is placed on the baseline of the content and horizontally centered.

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

```
\begin{fgimagebox}[⟨key=value pairs⟩]{⟨image filename⟩}
   ⟨content⟩
\end{fgimagebox}
```

```
\begin{fgimagebox*}[⟨key=value pairs⟩]{⟨image filename⟩}
   ⟨content⟩
\end{fgimagebox*}
```

Standalone macro and environment versions of the `fgimage` and `fgimage*` keys.

```
background=⟨background content⟩
background={⟨key=value pairs for image⟩}{⟨background content⟩}
```

Adds the given content as background layer to the main content. The background content is stretched to fit the size of the main content. It is also possible to provide `\adjustbox` or `\includegraphics` keys to modify the background content (before the resizing is done).

Examples:

```
\adjustbox{background={\color{red}xxx}}{Some text}
```

Some text

---

background*=⟨*background content*⟩
background*={⟨*key=value pairs for image*⟩}{⟨*background content*⟩}

Like `background` but does not resizes the background content. Instead it is added in its natural size but can be adjusted using `\adjustbox` or `\includegraphics` keys.

`\backgroundbox`[⟨*key=value pairs*⟩]{⟨*background content*⟩}
`\backgroundbox`*[⟨*key=value pairs*⟩]{⟨*background content*⟩}

\begin{`backgroundbox`}[⟨*key=value pairs*⟩]{⟨*background content*⟩}
  ⟨*content*⟩
\end{`backgroundbox`}

\begin{`backgroundbox*`}[⟨*key=value pairs*⟩]{⟨*background content*⟩}
  ⟨*content*⟩
\end{`backgroundbox*`}

Standalone macro and environment versions of the `background` and `background*` keys.

---

foreground=⟨*foreground content*⟩
foreground={⟨*key=value pairs for content*⟩}{⟨*foreground content*⟩}

Adds the given content as foreground layer to the main content. The foreground content is stretched to fit the size of the main content. It is also possible to provide `\adjustbox` or `\includegraphics` keys to modify the foreground content (before the resizing is done).

Examples:

```
\adjustbox{foreground={\color{red}xxx}}{Some text}
```

Some text

---

foreground*=⟨*foreground content*⟩
foreground*={⟨*key=value pairs for content*⟩}{⟨*foreground content*⟩}

Like `foreground` but does not resizes the foreground content. Instead it is added in its natural size but can be adjusted using `\adjustbox` or `\includegraphics` keys.

```
\foregroundbox[⟨key=value pairs⟩]{⟨foreground content⟩}
\foregroundbox*[⟨key=value pairs⟩]{⟨foreground content⟩}
```

```
\begin{foregroundbox}[⟨key=value pairs⟩]{⟨foreground content⟩}
    ⟨content⟩
\end{foregroundbox}
```

```
\begin{foregroundbox*}[⟨key=value pairs⟩]{⟨foreground content⟩}
    ⟨content⟩
\end{foregroundbox*}
```

Standalone macro and environment versions of the `foreground` and `foreground*` keys.

4.11 Density / Pixel size

The following keys allow to adjust the effective length of `pdftex` special px (pixel) unit. This can be required for images which do not have the correct DPI meta-data included.

`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}}%
}
```
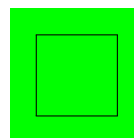
`pxdim`=⟨*length*⟩

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}}}
```

4.12 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+@%^&}_+!%
}
```
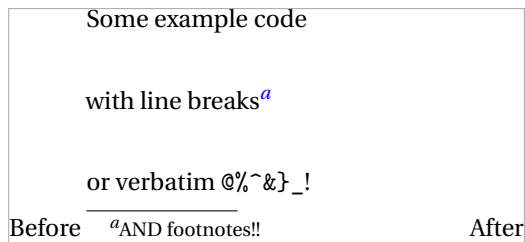

44

```
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
```

> Some example code
>
> with line breaks[a]
>
> or verbatim @%^&}_!
> ——————
> Before [a]AND footnotes!!   After

---

varwidth=⟨*width*⟩

Adds a `varwidth` environment from the `varwidth` package around the content. This is similar to the `minipage` environment but the final width of the content will be as large as the widest line in the content up to the maximum amount given by the width argument. This environment key is vert useful if the content produces some paragraph material which can't be stored in an horizontal box directly.

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

45

```
\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!

4.13 Floats and non-float replacements

The following keys can be used to turn an adjusted box into a float or a non-floating replacement which also allows for a caption. These keys must be used last and no other normal keys must be used afterwards, otherwise an error will occur because the added floating environment is boxed again.

> **caption**=⟨*caption text*⟩
> **caption**=[⟨*short caption*⟩]{⟨*long caption*⟩}

Defines a caption which will be used by a following `figure`, `float` or `nofloat` key. The position of the caption defaults to top for a `table` (non-)float and bottom for every other type.

> **captionabove**=⟨*caption text*⟩
> **captionabove**=[⟨*short caption*⟩]{⟨*long caption*⟩}

Like the `caption` key but forces the placement of the caption above the content. The vertical skip above and below a normal below caption are reversed in this case to get proper spacing. However, this is not done if the `caption` package is loaded. It is recommended to load this package with the `tableposition=above` option.

> **captionbelow**=⟨*caption text*⟩
> **captionbelow**=[⟨*short caption*⟩]{⟨*long caption*⟩}

Like the `caption` key but forces the placement of the caption below the content.

> **label**=⟨*label*⟩

Defines a label which will be used by a following `figure`, `float` or `nofloat` key.

> **figure**
> **figure**=⟨*placement*⟩

Turns the adjusted box into a `figure`. A previously used `caption` and `label` will be added. This is a specialised version of `float`={figure}.

> **float**=⟨*float type*⟩
> **float**={⟨*float type*⟩}[⟨*position*⟩]

Turns the adjusted box into a float of the given type, i.e. `figure`, `table` or any other custom defined float. A previously used `caption` and `label` will be added.

> **nofloat**=⟨*float type*⟩

This will add a non-floating replacement of the given float type (`figure`, `table`, etc.). The will place the content always at the current position which can lead to bad page-breaking. A caption will be in the same format as for real floats of the same type. A previously used `caption` and `label` will be added.

```
\begin{adjnofloat}{⟨float type⟩}
  ⟨content⟩
\end{adjnofloat}
```

This environment is used internally by `nofloat` to create a non-floating replacement of the given float type (`figure`, `table`, etc.). It can also be used directly with other code or be redefined to change the behavior of `nofloat`. Any redefinition should include \adjbox@nofloatcaptionsetup{#1} to set the caption type. The environment will be used in plain form, i.e. **\adjnofloat** ... **\endadjnofloat**, so if a group is required in a redefinition it should be added manually using **\begingroup** and **\endgroup**.

4.14 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. This key sets the size of the content before adding the code, i.e. keys like **width** or **scale** are executed on the content beforehand.

Addcode={⟨*code before*⟩}{⟨*code afters*⟩}

Identical to **addcode** but does not sets the size of the content first.

---
appcode=⟨*code afterwards*⟩

Appends the given LATEX code after the currently accumulated content. Note that this is the original content of the adjusted box *and* all code added by the previous keys.

It is recommended to wrap complex code into braces.

---
precode=⟨*code before*⟩

Prepends the given LATEX code before the currently accumulated content. Note that this is the original content of the adjusted box *and* all code added by the previous keys. The content is wrapped into curly braces, so the given code can use it as a macro argument. This key sets the size of the content before adding the code, i.e. keys like **width** or **scale** are executed on the content beforehand.

It is recommended to wrap complex code into braces.

Precode={⟨*code before*⟩}

Identical to **precode** but does not sets the size of the content first.

---
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 4.12) 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.

4.15 Change or discard content

```
phantom
phantom=h
phantom=v
```

This key replaces the content with an empty phantom box of the same dimension. If the "h" or "v" value is used the box will only have the same horizontal size or vertical sizes (height and depth) while using zero for the other dimension(s). For this the standard LaTeX macros `\phantom`, `\hphantom` and `\vphantom` are used internally.

```
\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

```
gobble
discard
```

These two keys are aliases and will discard the content after it was fully processed. Any keys used after one of this keys will operate on an empty content. These keys can be used in combination of the `gstore ...` keys to only measure but not typeset some content.

```
content=⟨new content⟩
```

This key discards the current content completely after it was fully processed and all earlier keys were applied and sets the content to the given ⟨new content⟩. This can be used to implement some draft box or censoring features.

Example:

```
\adjustbox{frame,rotate=-30,gstore sizes=\somelengtha{}{}\somelengthb,
    content={Censored!},rotate=30,totalheight=\somelengtha,width=\somelengthb,frame]
        {The real content}
```

4.16   Store box content

```
gstore=⟨\boxregister⟩
gstore*=⟨\boxregister⟩
```

These keys globally store the current content into the given box register. With the normal version the content is still preserved and can be further adjusted. With the starred version the content will be consumed and further keys will be applied to an empty content.

```
gstore width=⟨\lengthregister⟩
gstore height=⟨\lengthregister⟩
gstore depth=⟨\lengthregister⟩
gstore totalheight=⟨\lengthregister⟩
```

These key globally store the width, height, depth and totalheight into a length register, respectively. The content will not be altered by these keys. There are no starred versions like for `gstore`. If the content should only be measured but not typeset the `gobble`/`discard` key can be used afterwards.

```
gstore sizes=⟨\lengthregister⟩⟨\lengthregister⟩⟨\lengthregister⟩⟨\lengthregister⟩
```

Stores all four size values (width, height, depth and totalheight) of the content in the given length registers. This key should be used if more than one dimension should be stored, because it is much more efficient than using multiple of the above keys. If a dimension is not required it can be replaced by an empty argument, i.e. '{}'. Trailing dimensions can be skipped altogether if they are not required.

4.17 Process content

process

This key processes the keys and content encountered so far (i.e. all keys to its left) and sets the dimension macros to the sizes of the result. It is only required for certain key combinations, mostly the ones taken from `graphicx`, which are otherwise processed together.

4.18 Experimental Keys

The following features are experimental and may not work correctly yet. At the moment the `storebox` package must be loaded manually for this keys.

> **\splitbox**{⟨*split width*⟩}{⟨*split height*⟩}{⟨*content*⟩}

This macro boxes the given content and splits it in multiple fragments of the given width and height. This is done row by row starting from the upper left corner. The last column and row can have a smaller size than requested. Every fragment is placed using **\splitboxcmd**{⟨*fragment box*⟩} which is empty by default but can be redefined to e.g. draw a frame around each fragment. After every row except the last the macro **\splitboxnewline** is inserted which defaults to \\, but can also be redefined freely. After the last row the macro **\splitboxlastnewline** is inserted which defaults to **\splitboxnewline**.

The **\splitbox** content is stored using **\storebox** from the `storebox` package and therefore the whole content should only be stored once in the output file if the format is supported by that package (currently only PDF).

> \begin{**splitbox**}{⟨*split width*⟩}{⟨*split height*⟩}
>   ⟨*content*⟩
> \end{**splitbox**}

Environment version of **\splitbox**.

> **split**={⟨*split width*⟩}{⟨*split height*⟩}

The **split** key can be used with **\adjustbox** and uses **\splitbox** internally.

> **\pagebreakbox**{⟨*content*⟩}

The **\pagebreakbox** macro will split the content into multiple parts so that if fits on the current page. If it is larger than the rest of the current page and the full next page it is broken again until the last part fits on a page. This doesn't take any baselines into account and text line in the content may be split in two. This might be compensated manually by redefining the **\pagebreakboxoffset** macro (default: '\ht\strutbox'), which adjust the vertical offset of the first part.

> \begin{**pagebreakbox**}
> \end{**pagebreakbox**}

Environment version of **\pagebreakbox**.

> **pagebreak**

The key version of **\pagebreakbox**. There should no be any further keys used afterwards, because they will interfere with the page breaks.

5 Defining own Keys

The following macros can be used to define own keys or redefine existing ones.

\newadjustboxkey{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}
\newadjustboxkey∗{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}

Defines the new adjustbox key with the given default value if no value is provided. The value can be accessed inside the code as #1 Without a default value an error is raised if the value is missing. An error is raised if the key is already defined.

The normal form will add the code to the internal token register just before the previous content which is wrapped inside braces: ⟨*code*⟩{⟨*previous content*⟩}. Subsequent keys will receive this as content. The code can therefore read the content as macro argument. If the dimension of the content need to be accessed it should be boxed first using **\collectbox**{⟨*code*⟩}. Then the dimension are available using the usual macros and the content can be typeset using **\BOXCONTENT**. See the collectbox for additional information.

The starred form simply executes the given code in the internal group without changing the token register. In this case the content is not directly accessible and all subsequent keys are not yet processed. This form is useful to locally change settings for the current **\adjustbox** macro or adjustbox environment.

\renewadjustboxkey{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}
\renewadjustboxkey∗{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}

Like **\newadjustboxkey** but will redefine an existing key. An error is raised if the key is not already defined.

\provideadjustboxkey{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}
\provideadjustboxkey∗{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}

Like **\newadjustboxkey** but will define the key only if it is not already defined.

\defadjustboxkey{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}
\defadjustboxkey∗{⟨*key*⟩}[⟨*default value*⟩]{⟨*code*⟩}

Like **\newadjustboxkey** but will always define the key independent if it is already defined or not.