

# tikzscale — Absolute resizing of TikZ pictures and PGF plots without scaling text\*

Patrick Häcker<sup>†</sup>

Released 2013/05/22

## 1 Introduction

When dealing with graphics, there are different scaling demands. For *absolute* scaling, a width and/or height is given. Opposed to that, for *relative* scaling, a horizontal and/or vertical scaling factor is needed. This package only is about absolute scaling of tikzpicture environments. The different absolute scaling demands and their solutions are shown in table 1.

The tikzscale package adds and improves certain forms of absolute scaling for TikZ and PGFPlots, respectively. These scaling methods are the ones which are most useful, maybe even the only ones which are needed. During the scaling, the text sizes and line widths are left unscaled, which avoids inconsistency and visual distraction. PGFPlots itself can scale absolutely, but an approximation is used to achieve that. The tikzscale package uses optimization algorithms and warns if the scaling is not exact.

Using tikzscale all relevant scaling methods share the same user interface with the well known `\includegraphics` command, enabling some of its features like automatic file extension detection for TikZ and PGFPlots, too. Furthermore, the `\includegraphics` command is improved to look-up relative paths in the correct subdirectory, if a  $\LaTeX$  project is organized in subdirectories.

Relative scaling methods are mostly useless, as the sizes of the used images are often arbitrary, either determined by some resolution for rastered images or some arbitrary unit vector size for vector images, TikZ and PGFPlots. For traditional images and TikZ pictures, only proportional scaling methods giving either a width or a height make sense, as otherwise they get heavily distorted if the original aspect ratio is changed. As PGFPlots can handle different aspect ratios and aspect ratios are normally not predefined for plots, its requirement is the opposite: Both width and height are needed to avoid getting arbitrary sizes. For some special plots, the axis ratio can be given, as well. These requirements lead to the marked blue colors in table 1.

---

\*This file describes version v0.2.6, last revised 2013/05/22.

<sup>†</sup>E-mail: pat\_h@web.de

Table 1: Absolute graphic scaling methods. If multiple methods are available, the most native one is shown. Methods which **approximate** the scaling are shown in orange text color. **Recommended** methods are shown in blue textcolor.

(a) Scaling with scaled text and line widths.

scale	Images	TikZ/PGFPlots
to width proportionally	<code>\includegraphics [width=<i>unit</i>]</code>	<code>\resizebox {<i>width</i>}{!}</code>
to width keeping height	<code>\resizebox {<i>width</i>}{\height}</code>	<code>\resizebox {<i>width</i>}{\height}</code>
to height proportionally	<code>\includegraphics [height=<i>unit</i>]</code>	<code>\resizebox {!}{<i>height</i>}</code>
to height keeping width	<code>\resizebox {\width}{<i>height</i>}</code>	<code>\resizebox {\width}{<i>height</i>}</code>
to width and height	<code>\includegraphics [width=<i>unit</i>,height=<i>unit</i>]</code>	<code>\resizebox {<i>width</i>}{<i>height</i>}</code>

(b) Scaling with unscaled text and line widths without tikzscale.

scale	Images	TikZ	PGFPlots
to width proportionally	–	–	<code>[width=<i>unit</i>]</code>
to width keeping height	–	–	–
to height proportionally	–	–	<code>[height=<i>unit</i>]</code>
to height keeping width	–	–	–
to width and height	–	–	<code>[width=<i>unit</i>,height=<i>unit</i>]</code>

(c) Scaling with unscaled text and line widths with tikzscale.

scale	Images	TikZ	PGFPlots
to width proportionally	–	<code>\includegraphics [width=<i>unit</i>]</code>	<code>\includegraphics [width=<i>unit</i>]</code>
to width keeping height	–	–	–
to height proportionally	–	<code>\includegraphics [height=<i>unit</i>]</code>	<code>\includegraphics [height=<i>unit</i>]</code>
to height keeping width	–	–	–
to width and height	–	–	<code>\includegraphics [width=<i>unit</i>,height=<i>unit</i>]</code>

## 2 Usage and Examples

Loading the `tikzscale` package without loading other packages, does not do anything useful.

### 2.1 TikZ

If the `tikzscale` and the `tikz` packages are loaded, the `\includegraphics` command can be used to input and scale a `tikzpicture` environment located in a separate file.


As an example create the following `.tex`-file.

```
\documentclass{minimal}
\usepackage{tikz}
\usepackage{tikzscale}
\begin{document}
  \includegraphics[width=0.5\linewidth]{linewidth.tikz}
\end{document}
```

Furthermore create the following `.tikz`-file and save it as `linewidth.tikz` in the same directory as the above `.tex`-file.

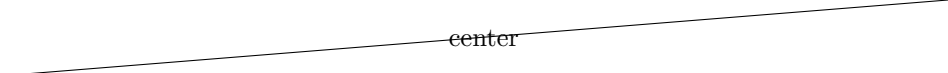
```
\begin{tikzpicture}
  \draw (0,0) - node {center} (\linewidth,1);
\end{tikzpicture}
```

The result of the compiled `.tex`-file should look like this.



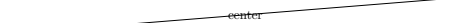
So although the original `tikzpicture` itself has the width of a complete line, it gets proportionally scaled down to half the width while being loaded from the `\includegraphics` command. Neither the line's thickness nor the text `center` are scaled. Compare the output to

```
\input{linewidth.tikz}
```



and

```
\resizebox{0.5\linewidth}{!}{\input{linewidth.tikz}}
```



to see `tikzscale`'s benefit.

### 2.2 PGFPlots

#### 2.2.1 Scaling of width and height

If the `pgfplots` package is loaded together with the `tikzscale` package, the user interface is the same. Instead of giving either a width or a height, both have to be given for `pgfplots`. Otherwise a default axis ratio is assumed (see section 2.2.2).

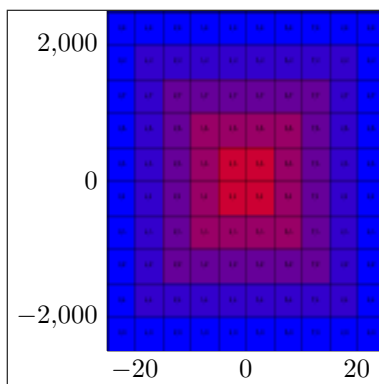


Figure 1: Using options `width=0.4\linewidth` and `height=0.4\linewidth` results in an overall quadratic graphic with overall width and height set to 40% of the linewidth.

So,

```
\input{pgfplots-test.tikz}
\begin{tikzpicture}\begin{axis}[width=3cm,height=2cm] ...
```

becomes

```
\includegraphics[width=3cm,height=2cm]{pgfplots-test.tikz}
\begin{tikzpicture}\begin{axis} ...
```

The benefit is a more accurate scaling algorithm, as the scaling with PGFPlots can be quite coarse. Another win is the unified interface, which simplifies the sharing of plots between projects enormously, as one file and thus one plot can be included in different projects with different sizes.

### 2.2.2 Scaling using axis ratio

The scaling described in the previous section scales the whole plot including all axis descriptions and legends to the given width and height. It can thus happen, that the plotted figure has a different size ratio than expected, if the x and y descriptions have different sizes as shown in figure 1. Sometimes, the x-axis and the y-axis should have a specific ratio, e.g. being equal, ignoring the axis description and other things. This is normally achieved by using PGFPlots' option `scale only axis`. Unfortunately, if this option would be used, a plot might be unsharable between two projects, if they have different requirements for the axis ratio. Thus, this option should not be used in such a case.

Instead, in `\includegraphics` there is a new option `axisratio` which must be used together with either `width` or `height`. It scales the whole plot including the axis description to the given width or height as in figure 2 while keeping the graphical part at a given axis ratio, where the ratio is defined by width divided

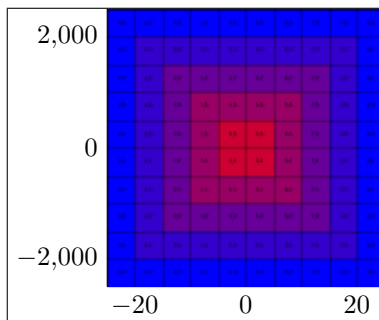


Figure 2: Using options `width=0.4\linewidth` and `axisratio=1` results in an quadratic graphic area with overall width set to 40% of the linewidth. The height follows from these constraints, so that the overall plot is not quadratic in general.

by height. The graphical part is thus not quadratic in general. If `axisratio` is omitted, i.e. only either height or width are given, it is assumed to be 1.

## 2.3 Hints for TikZ and PGFPlots

The whole `tikzpicture` environment must be in a separate file. This allows sharing of graphics between different  $\text{T}_\text{E}_\text{X}$  projects and a unified user interface via `\includegraphics`. Having `tikzpicture` environments directly in a `.tex`-file is not supported, i.e. they do not benefit from the `tikzscale` package. Multiple `tikzpicture` environments in one `.tikz`-file are not supported, either. Put things which always belong together in a shared `tikzpicture` environment and things which might be used separately in the future in separate files for code sharing across projects. The file ending may be omitted in the `\includegraphics` command, if it is one of `.tikz`, `.TIKZ`, `.TikZ`, `.pgf` or `.PGF`. At the moment, use only *either* width *or* height for normal (i.e. non-PGFPlots) `tikzpicture` environments and use width *and* height or one of both optionally together with `axisratio` for `tikzpicture` environments containing a PGFPlots' axis environment.

## 2.4 currfile

If the `tikzpicture` package is loaded together with the `currfile` package, another feature is activated. Suppose you have your project organized in the following directory tree with `directories` shown in blue color:

```
projectDirectory
  main.tex
  firstChapter
    firstChapter.tex
    firstGraphicOfFirstChapter.jpeg
    secondGraphicOfFirstChapter.tikz
  secondChapter
```

```
secondChapter.tex
firstGraphicOfSecondChapter.tikz
secondGraphicOfSecondChapter.jpeg
```

Further suppose the chapter.tex files are `\input`d in main.tex. Calling `\includegraphics{firstGraphicOfFirstChapter.jpeg}` in firstChapter.tex normally does not work. The reason is that the `\input{firstChapter.tex}` command in main.tex copies the content of firstChapter.tex into main.tex, so when the `\includegraphics` command is called, it is called from within project-Directory, thus the relative path lookup of firstGraphicOfFirstChapter.jpeg fails. Instead the command

```
\includegraphics{firstChapter/firstGraphicOfFirstChapter.jpeg}
```

can be used (example for a Unix system), but this is tedious and counter-intuitive.

If both tikzscale and currfile are loaded, the limitation is fixed, so that both `\includegraphics` commands succeed. Note, that this functionality supports the traditional graphic formats, too, and is also available without loading the TikZ or PGFPlots packages, although the package's name might imply otherwise.

## 3 Compatibility

### 3.1 Load Order

There is no constraint regarding the load order known, yet. TikZ, PGFPlots and currfile might all be loaded or not in all possible combinations and orders before or after tikzscale.

Using both the externalization library and tikzscale seems to have a race condition when a makefile is used with multiple jobs (`-jX` with  $X > 1$ ). The probability of getting errors increases with the number of jobs. For  $X = 1$ , obviously, no race condition could be observed. You should either avoid using mode *list and make* or have only one job if you want to be on the safe side.

Using `\tikzexternalenable` or `\tikzexternaldisable` inside of a tikzpicture leads to undefined behaviour when using tikzscale. It's not clear, what the correct behaviour should be and what the externalization library does without tikzscale.

Note, that there was a [bug](#) in the externalization library, which has been fixed on 25th of December in 2012, so you might want to use a more recent version of TikZ or PGFPlots.

### 3.2 Externalization library

TikZ' externalization library is supported. Its use is highly recommended, as tikzscale renders some graphics multiple times to get the correct size. The savings by using the externalization library can thus be huge.

### 3.3 Fitting library

Due to a [known bug in the fitting library](#), nodes with a `fit` option also need a `transform shape` option in order to be scalable. If they are not scalable, they normally do not contain the nodes as specified when `tikzscale` is used.

## 4 Further Ideas

- With careful considerations, it should be possible to reduce the average number of needed figure renderings, which should speed-up runs with very complicated figures.
- Add the external file optimization loop to `axis ratio` and `pgfplots`, as well.
- It might be a good idea to use the file names as figure names, but probably only if the name was not already set by the user. Additionally, there must be taken care to not try to write into a directory where there is no write access (e.g. reading a graphic from a system wide TeX installation)
- if graphic files are located in a subdirectory, the externalized files should also be in that subdirectory.
- allow in-file graphics by redefining the `tikzpicture` environment and accepting `tikzscale` and `tikz` options. The `tikzscale` options are evaluated using key filtering (`tikz` library) and the `tikz` options are forwarded.
- the package can test if a `pgfplot` is used (needed if normal TikZ graphics should be stretchable) by changing `\tikzscale@width` and or `\tikzscale@height` and measuring. If nothing changes, it must be a normal `tikzpicture` (the argument does not hold the other way round).
- it may be better to use the [depth as well](#)
- The final sizing parameters should be saved per figure in the aux file. The first rendering each run should be performed with the aux file's parameters into an `sbox`. The scaling algorithms should only be called, if the sizing requirements are not met. The purpose is similar to the externalization library.
- Using something like `[x=5pt]` as an argument to the `axis` environment, e.g. to scale the units in bar plots, is problematic, as `tikzscale` changes the behaviour, i.e. stops the scaling.

## 5 Contributions

- Jake
  - Encouraged the author to create this package.

- Dr. Christian Feuersänger
  - Encouraged the author to create this package and created PGFPlots.
  - Answered many questions and had a lot of good ideas regarding the externalization and beyond.
  - Fixed problems in the externalization library when used with tikzscale.
- David Carlisle
  - Created the `\xcmd` macro for this package, which is used in the documentation.
- Prof. Kai Arzheimer
  - Reported a bug when not using TikZ without PGFPlots, which lead to a fix.
  - Reported a bug that a non-existent macro is used, which lead to a fix.
- devendra
  - Reported bugs when using the externalization library together with tikzscale, which lead to a fix.
  - Reported a problem when using data files, which lead to a fix regarding `\endlinechar`.
- Mohammad Reza Keshtkaran
  - Reported a bug when using plain old L<sup>A</sup>T<sub>E</sub>X with an eps file, which lead to a fix.
  - Reported another bug when using plain old L<sup>A</sup>T<sub>E</sub>X, which lead to some rework to fully support L<sup>A</sup>T<sub>E</sub>X without additional code.
  - Reported a bug when using `\graphicspath`, which lead to a fix.
  - Reported the bug when using `\graphicspath` again, which lead to a correct fix even if `currfile` is not used.
- Andreas Tharang
  - Reported that the beamer class is incompatible with tikzscale, which lead to a change in tikzscale to fix this incompatibility.
  - Reported that the fitting library is incompatible with tikzscale due to a bug in the fitting library, which lead to a note in the documentation.
  - Created tests to improve the compatibility between beamer and tikzscale, which lead to support of Beamer's `\pause` command.
- Klaus Pribil
  - Reported an incompatibility with the pdfpages package, which lead to a fix in tikzscale.



- Christoph Schmidpeter
  - Reported a problem when accidentally adding a superfluous space into the graphics path, which lead to a detection and fix of that case in `tikzscale`.
- Jose Hissa Ferreira
  - Reported a bug when using a graphics path with multiple path entries, which lead to a fix.

## 6 Implementation

The basic idea is to first get the correct file name (i.e. find the path and the file extension), then determine the graphic type (i.e. TikZ or something else) and call either the original `includegraphics` command or the `tikzscale` command. Tikzpictures are then plotted into an invisible box and their size is measured. If their measured size differs from the requested size, they are replotted with corrected parameters to get the requested size. The correctly sized plots are then really plotted.

This command draws the plot's border at the right text border, so that thick points or label descriptions can reach into the margin. This should be limited to PGFPlots only if activated.

With the option below, the labels can be moved a bit to the left so that they reach to the text margin. `yticklabel style=align=right,inner sep=0pt,xshift=-0.1cm`

`\pgfmathsetglobalmacro` This is a general command, which might be useful for inclusion into the tikz package. It works similar to `\pgfmathsetglobalmacro` but has global scope.

```
1 \def\pgfmathsetglobalmacro#1#2{%
2 \pgfmathparse{#2}%
3 \global\let#1\pgfmathresult%
4 }
```

`\ifTikzLibraryLoaded` This is a general command, which might be useful for inclusion into the tikz package. This is taken from [stackexchange](#) and simplified.

```
5 \def\ifTikzLibraryLoaded#1#2#3{%
6 \ifcsdef{tikz@library@#1@loaded}{%
7 #2%
8 }{%
9 #3%
10 }%
11 }
```

`\ifExternalizationLoaded`

```
12 \def\ifExternalizationLoaded#1#2{%
13 \ifTikzLibraryLoaded{external}{#1}{#2}%
14 }
```

`\ifedefequal` `\ifedefequal{<first expression>}{<second expression>}{<true>}{<false>}` This is a general command, which might be useful for inclusion into the etoolbox package. It executes the true code if both expressions expand to the same and otherwise the false code. This test is often wanted, as the order of the arguments is not important and it does not matter if the user saves a value in another macro without expanding it.

```
15 \def\ifedefequal#1#2{%
16 \edef\etoolbox@ifedefequal@first{#1}%
17 \edef\etoolbox@ifedefequal@second{#2}%
18 \ifdequal{\etoolbox@ifedefequal@first}{\etoolbox@ifedefequal@second}%
19 }
```

`\edocsvlist` This is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\docsvlist` but expands its argument similar to `\def` vs. `\edef`, which is useful if the list is stored in a macro/variable.

```
20 \def\edocsvlist#1{%
21 \edef\tikzscale@edocsvlist{#1}%
22 \expandafter\docsvlist\expandafter{\tikzscale@edocsvlist}%
23 }
```

`\eforcsvlist` This is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\forcsvlist` but expands its argument similar to `\def` vs. `\edef`, which is useful if the list is stored in a macro/variable.

```
24 \def\eforcsvlist#1#2{%
25 \edef\tikzscale@eforcsvlist{#2}%
26 \expandafter\forcsvlist\expandafter{\expandafter#1\expandafter}\expandafter{\tikzscale@eforcsvlist}%
27 }
```

`\forgrouplist` This is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\forcsvlist` but uses TeX groups to separate elements instead of a comma separated list.

```
28 \def\forgrouplist#1#2{%
```

Use `\grouplistbreak` instead of `\forcsvlist`'s `\listbreak`, because the function given in the first argument can contain a call to `\listbreak`. In this case `\listbreak` is executed, although no break has been called, which lead to an error in the program, if `\listbreak` were used.

```
29 \def\grouplistbreak{\def\breakFor{}}%
30 \tikzscale@forGroupListElement{#1}#2\tikzscale@endList%
```

Delete `\breakFor` in case it has been set.

```
31 \undef\breakFor
32 }
```

`\tikzscale@forGroupListElement`

```
33 \NewDocumentCommand{\tikzscale@forGroupListElement}{mgu{\tikzscale@endList}}{%
```

Only do list processing if `\listbreak` has not been called.

```
34 \ifundef{\breakFor}{%
35 \IfValueTF{#2}{%
36 #1{#2}%
37 \tikzscale@forGroupListElement{#1}#3\tikzscale@endList%
38 }{%
39 #1{#3}%
40 }%
41 }{}}%
42 }
```

`\eforgrouplist` This is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\forgrouplist` but expands its argument similar to `\def` vs. `\edef`, which is useful if the list is stored in a macro/variable.

```
43 \def\eforgrouplist#1#2{%
44 \edef\tikzscale@grouplist{#2}%
45 \expandafter\forgrouplist\expandafter{\expandafter#1\expandafter}\expandafter{\tikzscale@grouplist}
46 }
```

`\tikzscale@trim` These is a general command to trim leading and trailing spaces, which might be useful for inclusion into another package taken from the following [homepage](#).

```
47 \def\tikzscale@trim#1{%
48 \ignorespaces#1\unskip
49 }
```

`\tikzscale@trimMacro` A possible present leading or trailing space in the macro's content is removed from the macro.

```
50 \def\tikzscale@trimMacro#1{%
51 \expandafter\IfBeginWith\expandafter{#1}{ }{%
52 \expandafter\StrGobbleLeft\expandafter{#1}{1}[#1]%
53 }{}}%
54 \expandafter\IfEndWith\expandafter{#1}{ }{%
55 \expandafter\StrGobbleRight\expandafter{#1}{1}[#1]%
56 }{}}%
57 }
```

`\elseif` This macro provides a conditional which supports an if with an arbitrary amount of elseif (none is also ok) and an optional else. With a simplified syntax (remove the tests and the grouping) this would be worth a separate package.

```
58 \NewDocumentCommand{\elseif}{mm}{%
59 \ifboolexpr{#1}{%
60 #2%
61 \elseif@absorb
62 }{%
63 \elseif@optional
64 }%
65 }
66 \NewDocumentCommand{\elseif@optional}{gg}{%
```

```

67 \IfValueTF{#1}{%
68 \IfValueTF{#2}{%
69 \ifboolexpr{#1}{%
70 #2%
71 \elseif@absorb
72 }{%
73 \elseif@optional
74 }%
75 }{%
76 #1%
77 }%
78 }{}%
79 }
80 \NewDocumentCommand{\elseif@absorb}{g}{%
81 \IfValueTF{#1}{%
82 \elseif@absorb
83 }{}%
84 }

```

T his command is from [Bruno Le Floch](#).

```

85 \ExplSyntaxOn
86 \NewDocumentCommand{\IfNoValueOrSplitEmptyTF}{mmm}{
87 \ifboolexpr{test {\IfNoValueTF{#1}} or test {\tl_if_eq:nnTF{#1}{}}}{
88 #2
89 }{
90 #3
91 }
92 }
93 \ExplSyntaxOff
94 %\end{macro}
95 %
96 %\begin{macro}
97 % The check \cmd{\tikzifexternalizehasbeencalled} from file tikzexternalshared.code.tex is not
98 % \begin{macrocode}
99 \def\tikzscale@ifExternalizationActive#1#2{%
100 \ifExternalizationLoaded{%
101 \ifdefequal{\tikz}{\tikzexternal@tikz@replacement}{%
102 #1%
103 }{%
104 \ifdefequal{\tikz}{\tikzexternal@origtikz}{%
105 }{%
106 \PackageWarning{tikzscale}{Status of externalization is unknown, thus I assume it is deactivated}
107 }%

```

It's important, that this code is below the above code, as the below code can change the meaning of `\tikz` through side effects.

```

108 #2%
109 }%
110 }{%
111 #2%

```

```

112 }%
113 }%

\tikzscale@debug Activate and deactivate debugging by commenting and uncommenting the following
code.
114 % \def\tikzscale@debug#1{%
115 % \PackageWarning{tikzscale}{#1}%
116 % }
117 \def\tikzscale@debug#1{}%

\activatetikzscale
118 \AtEndPreamble{%
Add the TikZ file extensions to the graphicx file extensions.
119 \def\tikzscale@tikzFileExtensions{.tikz,.TIKZ,.TikZ,.pgf,.PGF}%
120 % \def\tikzscale@tikzFileExtensions{.tikz,.TIKZ,.TikZ,.pgf,.PGF,.tex,.TEX}%
121 \DeclareGraphicsExtensions{\tikzscale@tikzFileExtensions,\Gin@extensions}%
Save the \includegraphics command.
122 \LetLtxMacro{\tikzscale@oldincludegraphics}{\includegraphics}%
Activate the enhanced includegraphics command at end of preamble, so that no
other package is interfering (besides on purpose).
123 \tikzscale@useEnhancedIncludegraphics
Also patch tikzpicture environment to temporarily deactivate the enhanced in-
cludegraphics command inside the tikzpicture environment in case the tikzpicture
environment is called directly (without includegraphics being called) and loading
another graphic (like a PNG file inside of a pgfplot).
124 \tikzscale@patchTikzpictureIncludegraphics
As \endtikzpicture does not seem to be redefined, patch it here (once) to ac-
tivate tikzscale's \includegraphics again. This is probably not necessary, but
might be handy if there are two tikzpicture environments in one includegraphics
environment.
125 \tikzscale@patchEndtikzpictureIncludegraphics
126 }

\tikzexternal
127 \AtEndPreamble{%
Activate the output of the graphics sizes into the dpth files (one file per graphic) if
externalization might be used (known at the end of preamble). This key is used if
the externalization library is activated to check if the scaling is correct, otherwise
the code is not needed.
128 \ifExternalizationLoaded{%
129 \pgfkeys{/pgf/images/external info}%
130 }{}%

```

Provide dummy commands, if the externalization library has not been loaded during the preamble.

```
131 % \ProvideDocumentCommand{\tikzsetnextfilename}{m}{}%
132 % \ProvideDocumentCommand{\tikzsetexternalprefix}{m}{}%
133 \ifpackageloaded{tikz}{%
```

Set a minimum accuracy `tikzscale` tries to achieve. TeX's accuracy is limited, thus, e.g. 0.04 pt, cannot always be achieved independent of the number of iterations. Use the `value` (0.1 pt in an experiment) which is used for overfull paragraph warnings, too.

```
134 \newlength{\tikzscale@accuracy}%
135 \setlength{\tikzscale@accuracy}{\hfuzz}%
```

This is needed in normal TikZ pictures and in PGFPlots, but as the `pgfplots` package loads the `tikz` package, it is fine to define it here.

```
136 \def\maxTestIterations{10}%
137 }{}
```

If the externalization library has been loaded, prepare it for use together with `tikzscale`.

```
138 \ifExternalizationLoaded{%
```

`\tikzexternaldisable` and `\tikzexternalenable` normally unintentionally deactivate the `tikzscale` commands (as they restore the original TikZ commands), so let them restore the `tikzscale` commands instead. The idea is to get the `tikzscale`'s `includegraphics` command being called and then redefine `tikzpicture` and do the rest of the work there. Do the patching always when `\tikzexternaldisable` or `\tikzexternalenable` is called, as the patching should also be done when `\includegraphics` is not used, but `\tikzpicture` is called directly.

```
139 \apptocmd{\tikzexternaldisable}{%
140 \tikzscale@useEnhancedIncludegraphics
141 \tikzscale@patchTikzpictureIncludegraphics
142 }{\PackageError{tikzscale}{Patching tikzexternaldisable failed}}%
143 %
```

```
144 \apptocmd{\tikzexternalenable}{%
145 \tikzscale@useEnhancedIncludegraphics
146 \tikzscale@patchTikzpictureIncludegraphics
147 }{\PackageError{tikzscale}{Patching tikzexternalenable failed}}%
148 % \end{macrocode}
```

149 % Patch the externalization command to also save the axis ratio if given. Unfortunately, `\cmd{`

```
150 % \begin{macrocode}
151 \LetLtxMacro{\tikzscale@externalend@storeshifts}{\pgf@externalend@storeshifts}%
152 \def\pgf@externalend@storeshifts#1{%
153 \tikzscale@externalend@storeshifts{#1}%
154 \ifpgfexternal@info
```

Write the axis ratio into the `dpth` file into variable `\tikzscale@oldAxisRatio`. The existence of the variable in the `dpth` file indicates if the axis ratio has been given in the last run.

```
155 \tikzscale@writeToDpth{#1}{\tikzscale@oldAxisRatio}{\requestedAxisRatio}%
```

```

156 \fi
157 }%
158 }-{}%
159 }

```

`\tikzscale@writeToDpth` Write the value of the third argument, if it exists, into the dpth file (write handle in the first argument) of the current figure, accessible by the second argument. The second and the third argument must contain macro names (including the backslash).

```

160 \def\tikzscale@writeToDpth#1#2#3{%
161 \ifdef{#3}{%
This is copied from the macro \pgf@externalend@storeshifts from file pgfcore-
external.code.tex.
162 \immediate\write#1{\noexpand\pgfexternal@restore{\noexpand\def\noexpand#2{#3}}}%
163 }-{}%
164 }

```

`\includegraphics`

```

165 \NewDocumentCommand{\tikzscale@includegraphics}{0{m}}-{}%

```

This command uses an empty optional argument for compatibility with the traditional `graphicx` command. Start a group, so that changed variables during processing the current `tikzpicture` due not influence other `tikzpicture`s. This is much more convenient, than resetting every single variable. Use `\beginngroup` instead of `\bgroup` to simplify finding unmatched braces.

```

166 \beginngroup

```

It happened at least once together with externalization, that the deactivation of the new `includegraphics` command did not work, so do it again to be safe (maybe reentrance problem with multiple `tikzpicture` calls?).

```

167 \LetLtxMacro{\includegraphics}{\tikzscale@oldincludegraphics}%

```

Do the patching of `endlinechar` and `tikzpicture` here, as `tikzpicture` should not be changed if not called via the new `\includegraphics` command.

```

168 \tikzscale@FixEndLine

```

Find the exact file name, as the ending and the path could be omitted.

```

169 \tikzscale@findExactFileName{tikzscale@fileName}{#2}%

```

Check if the found file is a TikZ file.

```

170 \tikzscale@isTikzFile{tikzscale@testTikzFile}{\tikzscale@fileName}%
171 \ifcsdef{tikzscale@testTikzFile}{%
172 \tikzscale@includetikz[#1]{\tikzscale@fileName}%
173 }-{}%

```

Restore `\endlinechar` before calling code from other packages. This is not only cleaner, but really avoids an error when using the plain old latex (with dvi output) with an eps graphic.

```

174 \tikzscale@restoreEndLineChar

```

```

175 \tikzscale@oldincludegraphics[#1]{\tikzscale@fileName}%
176 }%
177 \endgroup
178 }%

```

`\tikzscale@useEnhancedIncludegraphics` Replace the `\includegraphics` command by `tikzscale`'s more generic command, to provide a consistent user interface.

```

179 \def\tikzscale@useEnhancedIncludegraphics{%
180 \LetLtxMacro{\includegraphics}{\tikzscale@includegraphics}%
181 }

```

`\tikzscale@FixEndLine` `tikzpicture` environment gets redefined: - without external library: only inside `tikzscale` (once) - with external library: additionally, whenever `\externalenable` or `\externaldisable` is called (`\tikzpicture` and `\endtikzpicture`) Use cases to patch `tikzpicture`: - to use `tikzscale`'s `includegraphics` - to restore end of line character Constraints: - The patches have to be applied at the beginning of `\tikzpicture` and the end of `\endtikzpicture`, as `\tikzpicture` might not be executed completely when using external, as then the content of the `tikzpicture` environment is not executed at all. - The patches should not accumulate - A group might make sense to have a local scope

```

182 \def\tikzscale@FixEndLine{%

```

There is a leading space character introduced by the externalization library, if the file is input directly. Thus use a trick to avoid that space. Furthermore, TikZ introduces with a specific version a trailing space character. To get rid of all space character issues, just solve the problem here once and for all. Note, that the redefinition of `\endlinechar` is local to the current group, so it does not have to be restored at the end of the group.

```

183 \edef\tikzscale@restoreEndLineChar{\endlinechar=\the\endlinechar\relax}%
184 \endlinechar=-1%

```

Restore the `\endlinechar` during the execution of the `tikzpicture` environment. This is necessary, for example, if data is read from a table and the data entries are separated by newline characters. Not restoring the `\endlinechar` would **distort the data**. Use `\apptocmd` to call the command inside the group opened by `tikzpicture`. Thus, nothing has to be done in `\endtikzpicture` regarding `\endlinechar`.

```

185 \tikzscale@addRestoreEndLineCharToTikzpicture
186 %
187 \apptocmd{\endtikzpicture}{%
188 \endlinechar=-1%
189 }{\PackageError{tikzscale}{Patching endtikzpicture failed}}%
190 }%

```

`RestoreEndLineCharToTikzpicture`

```

191 \def\tikzscale@addRestoreEndLineCharToTikzpicture{%
192 \pretocmd{\tikzpicture}{%
193 \tikzscale@restoreEndLineChar
194 }{\PackageError{tikzscale}{Patching tikzpicture failed}}%
195 }

```



patchTikzpictureIncludegraphics

```
196 \def\tikzscale@patchTikzpictureIncludegraphics{%
197 % Deactivate the new includegraphics command inside of tikzpictures, as a tikzpicture might lo
198 % \begin{macrocode}
199 \pretocmd{\tikzpicture}{%
200 \LetLtxMacro{\includegraphics}{\tikzscale@oldincludegraphics}%
201 }{\PackageError{tikzscale}{Patching tikzpicture failed}}%
202 %
203 }
```

chEndtikzpictureIncludegraphics

```
204 \def\tikzscale@patchEndtikzpictureIncludegraphics{%
205 \apptocmd{\endtikzpicture}{%
206 \LetLtxMacro{\includegraphics}{\tikzscale@includegraphics}%
207 }{\PackageError{tikzscale}{Patching endtikzpicture failed}}%
208 }
```

\tikzscale@findExactFileName

Find the exact file name of a graphic file by testing several paths and file endings if there are degrees of freedom. The file name is saved in the command sequence name given by the first argument.

```
209 \NewDocumentCommand{\tikzscale@findExactFileName}{mm}{%
```

Delete the return variable if it already exists to allow checking if a file has been found.

```
210 \csundef{#1}%
```

Create a helper function used inside the file ending evaluation.

```
211 \def\tikzscale@checkDirectory##1{%
212 \def\tikzscale@checkExtension####1{%
213 \IfFileExists{##1#2####1}{%
```

Use \csedef instead of \csdef here, to be completely sure to only have a string left. This avoids problems when using tikzscale together with the pdfpages package and should generally be the right thing.

```
214 \csedef{#1}{##1#2####1}%
```

Break the inner (\forcsvlist) loop over file extensions.

```
215 \listbreak
216 }{)%
217 }%
```

Test all possible file extensions and do not forget that the extension might already be given. \Gin@extensions returns the **current content** set by \DeclareGraphicsExtensions.

```
218 \forcsvlist{\tikzscale@checkExtension}{}, \Gin@extensions}%
219 \ifcsdef{#1}{%
```

Break the outer (\forgrouplist) loop over directories.

```
220 \grouplistbreak
221 }{)%
222 }%
```

Set the graphics path, to also find graphics in the last (current) input directory or in completely separate paths. Set it here to get updates if the user uses the `\graphicspath` command inside of the document body.

```

223 \tikzscale@setGraphicsPath
224 \eforgrouplist{\tikzscale@checkDirectory}{\tikzscale@graphicspath}%
If no file has been found, return the given file name, as includegraphics should try
its best.
225 \ifcsundef{#1}{%
226 \csdef{#1}{#2}%
227 }{}%
228 }

```

`\tikzscale@setGraphicsPath` The `\graphicspath` command is used to set additional directories, which are searched for graphics. `\Ginput@path` is used to get the **current content**.

```

229 \NewDocumentCommand{\tikzscale@setGraphicsPath}{}{}%
Remove possible leading or trailing spaces in the graphics path, as they lead to ugly
string output before printing the graphic. Inserting such a space in the graphics
path is a user's error, but it can happen easily as not all users are aware of TeX's
newline issues. Fix the original path variable and not only tikzscale's variable, as
this seems to be a general problem.
230 \ifdef{\Ginput@path}{%
231 \tikzscale@trimMacro{\Ginput@path}%
232 }{}%
233 \ifdef{\currfiledir}{%
234 \ifdef{\Ginput@path}{%
235 \def\tikzscale@graphicspath{\currfiledir\Ginput@path}}%
236 }{}%
237 \def\tikzscale@graphicspath{\currfiledir}}%
238 }%
239 }{}%
240 \ifdef{\Ginput@path}{%
241 \def\tikzscale@graphicspath{\Ginput@path}}%
242 }{}%
243 \def\tikzscale@graphicspath{}}%
244 }%
245 }%
246 }%

```

`\tikzscale@isTikzFile` The first argument is the macro name (without backslash), which gets defined if the file is a tikzfile. The second argument is the file name.

```

247 \NewDocumentCommand{\tikzscale@isTikzFile}{mm}{}%
Create a helper function used inside the evaluation.
248 \def\do##1{%
249 \IfEndWith{#2}{##1}{%
250 \csdef{#1}{}%
251 \listbreak
252 }{}%

```

```

253 }%
Delete macro so that defining it is really indicating something.
254 \csundef{#1}%
255 \edocsvlist{\tikzscale@tikzFileExtensions}%
256 }

```

`\pgfkeys` This is **similarly** done.

```

257 \pgfkeys{
258 /tikzscale/.is family, /tikzscale,
259 width/.code = {\pgfmathsetmacro{\requestedWidth}{#1}},
260 width/.value required,
261 height/.code = {\pgfmathsetmacro{\requestedHeight}{#1}},
262 height/.value required,
263 axisratio/.code = {\pgfmathsetmacro{\requestedAxisRatio}{#1}},
264 axisratio/.value required
265 }

```

```

\tikzscale@includetikz \tikzscale@includetikz{<filename>}
\tikzscale@includetikz[<width=1cm>]{<filename>}
\tikzscale@includetikz[<height=1cm>]{<filename>}
\tikzscale@includetikz[<height=1cm,width=1cm>]{<filename>}
\tikzscale@includetikz[<width=1cm,height=1cm>]{<filename>}

```

This command allows the inclusion of a tikz file like a graphics file. Thus instead of writing `\includegraphics[width=\linewidth]fileWithoutEnding` write `\tikzscale@includetikz[width=\linewidth]fileWithoutEnding` If only one of width or height are given, scale proportionally to fulfill the requirement. If both are given, scale non-proportionally to required width and height. Therefore, for normal tikzpictures only give either width or height, as the aspect ratio is already determined by the coordinate limits in the tikzpicture, but give width and height for PGFPlots, as the aspect ratio is unknown for these plots. `\NewEnviron` could be used to handle something like `verbose` in a tikzpicture, but at the moment, this is unsupported. The used code is the same as the uncommented code, but also **compatible with class beamer**.

```

266 % \NewDocumentCommand{\tikzscale@includetikz}{0{m}}{%
267 \newcommand{\tikzscale@includetikz}[2] []{%

```

Check the keys here already, as they are needed both to see if already externalized files fulfill their requirements and to handle unexternalized files.

```

268 \pgfkeys{/tikzscale, #1}%

```

Check if the current graphic should be either drawn and scaled or simply included. As externalization can get activated or deactivated at any time (if the library has been loaded in the preamble), check in every call what to do.

```

269 \tikzscale@ifExternalizationActive{%

```

If externalization library has been loaded and is active, draw and scale the graphic if it is to be externalized.

```

270 \tikzifexternalizingnext{%

```

```

271 \tikzscale@debug{Externalizing the file #2}%
272 \tikzscale@includetikzUnexternalized{#2}%
273 }{%
274 \tikzscale@includetikzWithExternalization{#2}%
275 }%
276 }{%

```

Always draw and scale the graphic if externalization library has not been loaded or is deactivated.

```

277 \tikzscale@includetikzUnexternalized{#2}%
278 }%
279 }

```

scale@includetikzUnexternalized

```

280 \def\tikzscale@includetikzUnexternalized#1{%
281 \elseif{test {\ifundef{\requestedWidth}} and test {\ifundef{\requestedHeight}} and test {\ifun
282 \tikzscale@debug{no option given}%

```

If no option is given, directly load the content, as nothing should get scaled.

```

283 \tikzscale@trim{\input{#1}}%
284 }{test {\ifdef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%
285 \tikzscale@debug{width and height given}%

```

If width and height are given, the content must be a pgfplot, so scale it. The plot currently only had approximately the given size without calling the `resizeTo` macro, due to a (known) bug in PGFPlots.

```

286 \tikzscale@resizePlotTo{#1}%
287 }{test {\ifdef{\requestedAxisRatio}}}{%
288 \tikzscale@debug{axis ratio given}%
289 \tikzscale@includeAxisRatio{#1}%
290 }{test {\ifundef{\requestedAxisRatio}}}{%
291 \tikzscale@debug{width or height given}%

```

Use this test as a check if PGFPlots has been loaded.

```

292 \ifdef{pgfplotsset}{%

```

If only either width or height is given it can be a normal tikzpicture or a plot with `axisratio=1`. Let's guess that it is a plot with default `axisratio`. If the guess is wrong, the called function detects that scaling the plot does not work and automatically calls `\tikzscale@includeNormalTikzpicture`.

```

293 \def\requestedAxisRatio{1}%
294 \tikzscale@includeAxisRatio{#1}%
295 }{%
296 \tikzscale@debug{no pgfplots loaded}%

```

If PGFPlots has not been loaded, it can only be a TikZPicture.

```

297 \tikzscale@includeNormalTikzpicture{#1}%
298 }%
299 }{%
300 % Everything else results in an error.
301 \tikzscale@invalidKeyError{#1}%

```

```

302 }%
303 }

\includetikzWithExternalization This macro includes a tikzpicture file using the externalization library. As a pre-
condition, the externalization must be loaded and active.
304 \NewDocumentCommand{\tikzscale@includetikzWithExternalization}{m}{%
Try to load a dpth file to get the sizes pgfexternalwidth and pgfexternalheight as
well as tikzscale@oldAxisRatio of the externalized graphic.
305 \tikzexternalgetnextfilename{\tikzscale@externalizationName}%
306 \pgfexternalreaddpth{\tikzscale@externalizationName}%
Check if the next figure has to be remade. If no dpth file exist, it need not and
must not be remade, as otherwise no md5-file is generated and thus one extra
compilation run is necessary.
307 \IfFileExists{\tikzscale@externalizationName.dpth}{%
308 \tikzscale@checkRequestedSizeChanges
309 }{}%

The figure is either remade or the PDF is included with the following call. The
former is correct if the file has been changed, the latter is correct if the last run
was correct and nothing has changed since then. The only case left is if the
figure has neither changed, nor is the exported size correct, as the file is then
regenerated repeating the same error as last time. To change something, the
new externalisation would need the result of the old externalization, but as the
externalization is processed in a separate LaTeX process, it's non-trivial to push
the information there.
310 \tikzscale@trim{\input{#1}}%
311 }

\tikzscale@checkRequestedSizeChanges
312 \NewDocumentCommand{\tikzscale@checkRequestedSizeChanges}{}{}%
Check if the sizes are still correct, i.e. agree with the sizes of the externalized PDF
graphic. The saved axis ratio from the last run is checked, too, as it might have
been changed by the user between the last run and the current run.
313 \ifdef{\requestedWidth}{%
314 \ifdef{\pgfexternalwidth}{%
315 \tikzscale@ifSizeDifference{\requestedWidth - \pgfexternalwidth}{%
316 \tikzset{external/remake next}%
317 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of width diff
318 % \tikzscale@warnIfSizeDifference{\requestedWidth}{\pgfexternalwidth}{current file}%

It would be possible to calculate a new size, if the old size did not fit, but this
information would be needed to push into the externalization process, what is
hard. The idea were requested = requested + measuredOld - fileSize.
319 }{}%
320 }{}%
321 \tikzset{external/remake next}%
322 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of no externa

```

```

323 }%
324 }{}%
325 \ifdef{\requestedHeight}{%
326 \ifdef{\pgfexternalheight}{%
327 \tikzscale@ifSizeDifference{\requestedHeight - \pgfexternalheight}{%
328 \tikzset{external/remake next}%
329 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of height dif
330 }{}%
331 }{}%
332 \tikzset{external/remake next}%
333 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of no externa
334 }%
335 }{}%
336 \ifdef{\requestedAxisRatio}{%
337 \ifdef{\tikzscale@oldAxisRatio}{%
338 \tikzscale@ifSizeDifference{\requestedAxisRatio - \tikzscale@oldAxisRatio}{%
339 \tikzset{external/remake next}%
340 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of axis ratio
341 }{}%
342 \undef{\tikzscale@oldAxisRatio}%
343 }{}%
344 \tikzset{external/remake next}%
345 \tikzscale@debug{Regenerate \tikzscale@externalizationName \MessageBreak because of no externa
346 }%
347 }{}%
348 }

```

`\tikzscale@preparePlot`

```

349 \NewDocumentCommand{\tikzscale@preparePlot}{}{}%

```

Set a scaling factor or a width and height for the plot, which will be loaded. The `\tikzset` and `\pgfplotsset` commands have local scope. The internal redefinition of the style is correct, because if one `tikzpicture` includes another one, the scaling factor is reset so that it does not get **squared** in the inner one. Note that if a user-defined style thus is ignored in this special case. The styles are defined here, so that files which are inputted without the `\includegraphics` command are not affected.

```

350 \pgfplotsset{every axis/.append style={width=\tikzscale@width,height=\tikzscale@height,every a
351 }
352 \NewDocumentCommand{\tikzscale@prepareTikzpicture}{}{}%
353 \tikzset{every picture/.style={scale=\tikzscale@scale,every picture/.style={}}}%
354 }

```

`\tikzscale@includeNormalTikzpicture`

```

\tikzscale@includeNormalTikzpicture{\file name}
355 \NewDocumentCommand{\tikzscale@includeNormalTikzpicture}{m}{}%
356 \tikzscale@prepareTikzpicture
357 \elseif{test {\ifdef{\requestedWidth}} and test {\ifundef{\requestedHeight}}}{%
358 \def\requestedSize{\requestedWidth}%
359 \tikzscale@scaleTikzpictureTo{\wd}{\tikzscale@trim{\input{#1}}}{#1}%

```

```

360 }{test {\ifundef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%
361 \def\requestedSize{\requestedHeight}%
362 \tikzscale@scaleTikzpictureTo{\ht}{\tikzscale@trim{\input{#1}}}{#1}%
363 }{%
364 \tikzscale@invalidKeyError{#1}%
365 }%
366 }

```

`\tikzscale@invalidKeyError`

```

367 \NewDocumentCommand{\tikzscale@invalidKeyError}{m}{%
368 \PackageError{tikzscale}{Invalid key for TikZ graphic}{Change key #1 into a valid key.}%
369 }

```

`\tikzscale@includeAxisRatio` `\tikzscale@includeAxisRatio{<file name>}`

```

370 \NewDocumentCommand{\tikzscale@includeAxisRatio}{m}{%
Try to set initial sizes close to the requested sizes, to improve the optimization's
speed.
371 \pgfplotsset{every axis/.append style={scale only axis,every axis/.style={}}}%
372 \elseif{test {\ifdef{\requestedWidth}} and test {\ifundef{\requestedHeight}}}{%
373 \let\requestedSize\requestedWidth
374 \def\tikzscale@width{\requestedWidth}%
375 \pgfmathsetmacro{\tikzscale@height}{\requestedWidth / \requestedAxisRatio}%
376 \tikzscale@resizePlotWithAxesRatioTo{\wd}{\tikzscale@width}{\tikzscale@trim{\input{#1}}}{#1}%
377 }{test {\ifundef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%
378 \let\requestedSize\requestedHeight
379 \def\tikzscale@height{\requestedHeight}%
380 \pgfmathsetmacro{\tikzscale@width}{\requestedHeight * \requestedAxisRatio}%
381 \tikzscale@resizePlotWithAxesRatioTo{\ht}{\tikzscale@height}{\tikzscale@trim{\input{#1}}}{#1}%
382 }{%
383 \tikzscale@invalidKeyError{#1}%
384 }%
385 }

```

`\tikzscale@scaleTikzpictureTo` `\scalteTo{<\wd or \ht>}{<to-be-scaled content>}{<file name>}` The first argument determines if a specific width or a specific height should be achieved by scaling.

```

386 \NewDocumentCommand{\tikzscale@scaleTikzpictureTo}{mmm}{%
387 \tikzscale@debug{At beginning scale, requestedWidth=\requestedWidth}%
Deactivate the externalization, as the measurements to determine the correct size
should not be externalized.
388 \tikzscale@conditionalDisableExternalization

```

When scaling a tikzpicture, only the drawings are scaled, but nodes are not scaled. So in general, there are horizontal or vertical areas, where the picture contains only unscaled nodes, and areas where the picture contains scalable drawings. Mathematically all scaled and all unscaled areas can be combined, so that there is one area of fixed size and one variable sized area. Thus scaling only by multiplication of a factor is incorrect in general. To do the correct scaling, the fixed

area size must be known. As there are two unknown parameters, i.e. fixed area size and variable area size, the fixed area size can be calculated by measuring the tikzpicture with two different scalings. A special scaling factor is used, to get the size close to the final size minimizing numerical and logical errors.

```
389 \def\tikzscale@scale{1}%
390 \tikzscale@measureSize{\measuredFirst}{#1}{#2}%
391 \pgfmathsetmacro{\tikzscale@scale}{\requestedSize/\measuredFirst}%
392 \tikzscale@measureSize{\measuredSecond}{#1}{#2}%
```

It can happen, that there are no variable areas. Furthermore, the original size could already fit. Avoid numerical problems in both cases by directly drawing the picture. Do not compare the float values directly, as TeX's precision is quite limited.

```
393 \tikzscale@ifSizeDifference{\measuredSecond - \requestedSize}{%
```

If a plot is not scalable (e.g. consisting of a node only), but is not correctly scaled, exit with an error.

```
394 \tikzscale@ifSizeDifference{\measuredFirst - \measuredSecond}{%
395 }{%
```

```
396 \PackageError{tikzscale}{Requested to scale unscalable graphic}{Do not set width or height for
397 }%
```

We know, that the variable sized area scales with the scaling factor, thus it holds  $\text{\scale} * \text{\variableFirst} = \text{\variableSecond}$ , with  $\text{\variableFirst} = \text{\measuredFirst} - \text{\fixedSize}$  and  $\text{\variableSecond} = \text{\measuredSecond} - \text{\fixedSize}$ , which can be solved by substitution and results in

```
398 \pgfmathsetmacro{\fixedSize}{(\tikzscale@scale*\measuredFirst - \measuredSecond) / (\tikzscale@
```

Now, to get the correct scaling factor, only take the variable areas into account, as it holds  $\text{\scaleFinal} = \text{\variableSizeFinal} / \text{\variableSizeOriginal}$  with  $\text{\variableSizeFinal} = \text{\requestedSize} - \text{\fixedSize}$  and  $\text{\variableSizeOriginal} = \text{\measuredFirst} - \text{\fixedSize}$ , which results in

```
399 \pgfmathsetmacro{\tikzscale@scale}{(\requestedSize - \fixedSize) / (\measuredFirst - \fixedSize)}
```

Additionally or alternatively the brute force approach to iteratively improve the solution can be used.

```
400 \foreach \l in {1,...,\maxTestIterations}{%
401 \tikzscale@measureSize{\measuredIntermediate}{#1}{#2}%
```

Optimize until the absolute difference is small enough, although the (relative) size ratios are used to calculate a new scaling factor.

```
402 \tikzscale@ifSizeDifference{\measuredIntermediate-\requestedSize}{%
```

First divide before multiply to avoid overflowing (at 16384).

```
403 \pgfmathsetmacro{\errorRatio}{\measuredIntermediate/\requestedSize}%
404 \tikzscale@debug{errorRatio=\errorRatio\MessageBreak for #3}%
405 \pgfmathsetglobalmacro{\tikzscale@scale}{\tikzscale@scale/\errorRatio}%
406 }{%
407 \breakforeach%
408 }%
409 }%
```



Do a last measurement to be able to warn if the size does not fit good enough. This measurement has to be done before possibly reactivating the externalization, as measurements with activated externalization can lead to wrong measurement results (possibly due to calling `\shipout` inside of the measurement). The assumption is, that the real graphic size does not change if the externalization gets activated, which all tests seem to confirm.

```

410 \tikzscale@measureSize{\measuredFinal}{#1}{#2}%
411 \tikzscale@warnIfSizeDifference{\measuredFinal}{\requestedSize}{#3}%
412 \tikzscale@testGraphicFileForRetry{#1}{#2}{#3}{\measuredFinal}%
413 }{%
414 \tikzscale@testGraphicFileForRetry{#1}{#2}{#3}{\measuredSecond}%
415 }%
416 }

```

`\tikzscale@testGraphicFileForRetry`

The macro tests the size of the generated graphic file. If the size is not as the requested size, the optimization is redone with an adapted optimization criterion to compensate the error done in the last run.

```

\tikzscale@testGraphicFileForRetry{(\wd or \ht)}{(to-be-scaled content)}{(file
name)}{(last measurement)}

```

```

417 \def\tikzscale@testGraphicFileForRetry#1#2#3#4{%

```

Reactivate externalization to prepare the figure to be really rendered.

```

418 \tikzscale@conditionalEnableExternalization{#3}%

```

Before externalizing the file, save the current externalization file name to be possibly used later. This code must be run after the reactivation of the externalization and before the rendering itself.

```

419 \tikzscale@ifExternalizationActive{%
420 \tikzexternalgetnextfilename{\tikzscale@externalizationName}%
421 }{%

```

Finally, externalize and include the graphic with the final size. The graphic must not be included by reusing the measuredSize-box, as this leads to a subtly wrong behaviour when generating the PDF files. It can be observed by running the test suite with externalization and checking that no file is regenerated in the second run. Reusing the box can also lead to compile errors, if the problematic graphic is the last graphic in the document.

```

422 #2%

```

If the externalization is active, it might happen, that a figure was scaled correctly according to the request, but the externalized figure has a different size. The reason for that behaviour is unknown. The only chance we have is to calculate the error, rewind everything and try again with the compensated error.

```

423 \tikzscale@ifExternalizationActive{%

```

Read the size of the saved PDF file and save it on `\pgfexternalsize`.

```

424 \pgfexternalreadpsth{\tikzscale@externalizationName}%
425 \tikzscale@ifIsWidth{#1}{%
426 \edef\pgfexternalsize{\pgfexternalwidth}%

```

```

427 }{%
428 \edef\pgfexternalsize{\pgfexternalheight}%
429 }%

```

Save the `\requestedSize` from the user in `\originalRequestedSize`, as the former value has to be overwritten for the next try. Do not use `\def`, as that would later be expanded to the current and not to the original value.

```

430 \ifundef{\originalRequestedSize}{%
431 \let\originalRequestedSize\requestedSize
432 }{%
433 \tikzscale@debug{originalRequestedSize=\originalRequestedSize, requestedSize=\requestedSize, p

```

If a size difference is left, compensate it, decrease the counter and try the scaling again.

```

434 \tikzscale@ifSizeDifference{\originalRequestedSize-\pgfexternalsize}{%
435 \pgfmathsetmacro{\requestedSize}{\requestedSize + #4 - \pgfexternalsize}%
436 \tikzscale@decreaseFigureCounter
437 \tikzscale@scaleTikzpictureTo{#1}{#2}{#3}%
438 }{%
439 }{%
440 }

```

`\tikzscale@decreaseFigureCounter`

Decrease the externalization figure counter. This way the last externalized figure will be overwritten when the next figure is externalized. This is useful if the same figure should be externalized again. This code is a modified copy of `\tikzexternal@getnextfilename@advancecount` from file `tikzexternal-shared.code.tex`.

```

441 \def\tikzscale@decreaseFigureCounter{%

```

Use a group, so that the used counter register is not changed outside of this macro. This is necessary, as the global figure counter is not saved in a counter as one might think, but in a macro, so it must be copied to a local counter in order to be easily modified.

```

442 \begingroup

```

Get the name of the macro holding the figure counter value.

```

443 \edef\macroName{c@tikzext@no@\pgfkeysvalueof{/tikz/external/figure name}}%

```

Use a TeX counter and store the figure counter value therein.

```

444 % \c@pgf@counta=\csname\macroName\endcsname\relax
445 \c@pgf@counta=\csuse{\macroName}\relax

```

Decrease the value of the local counter.

```

446 \advance\c@pgf@counta -1\relax

```

Save the value of the local counter back to the global counter macro.

```

447 \expandafter\xdef\csname\macroName\endcsname{\the\c@pgf@counta}%
448 \endgroup
449 }%

```

```

\tikzscale@resizePlotTo \tikzscale@resizePlotTo{\file name}
450 \NewDocumentCommand{\tikzscale@resizePlotTo}{m}{%
451 \def\fileName{#1}%
452 \def\content{\tikzscale@trim{\input{#1}}}%
453 \tikzscale@preparePlot
454 \def\tikzscale@width{\requestedWidth}%
455 \def\tikzscale@height{\requestedHeight}%

```

Deactivate the externalization, as the measurements to determine the correct size should not be externalized.

```

456 \tikzscale@conditionalDisableExternalization

```

Improve the solution iteratively until it is good enough.

```

457 \foreach \l in {1,...,\maxTestIterations}{%

```

Using the box allows measuring the width and height with one rendering run.

```

458 \sbox{\tikzscale@measuredSize}{\content}%

```

Determine the remaining error and check if it is larger than a threshold.

```

459 \pgfmathsetmacro{\widthDifference}{\wd\tikzscale@measuredSize - \requestedWidth}%
460 \pgfmathsetmacro{\heightDifference}{\ht\tikzscale@measuredSize - \requestedHeight}%

```

Output error in current iteration for debugging.

```

461 % widthDifference: \widthDifference, heightDifference: \heightDifference\\% Debugging

```

Check if the remaining error is larger than a threshold.

```

462 \ifboolexpr{test {\tikzscale@ifSizeDifference{\widthDifference}} or test {\tikzscale@ifSizeDif

```

Correct the dimension by the error. Use a global assignment, as each iteration in the loop is put into a separate group.

```

463 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@width - \widthDifference}%
464 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@height - \heightDifference}%
465 }{%
466 \breakforeach
467 }%
468 }%

```

Do a last measurement to be able to warn if the size does not fit good enough. This measurement has to be done before possibly reactivating the externalization, as measurements with activated externalization can lead to wrong measurement results (possibly due to calling `\shipout` inside of the measurement). The assumption is, that the real graphic size does not change if the externalization gets activated, which all tests seem to confirm.

```

469 \sbox{\tikzscale@measuredSize}{\content}%
470 \tikzscale@warnIfSizeDifference{\requestedWidth}{\wd\tikzscale@measuredSize}{\fileName's width
471 \tikzscale@warnIfSizeDifference{\requestedHeight}{\ht\tikzscale@measuredSize}{\fileName's heigh

```

Finally, externalize and include the graphic with the final size. The graphic must not be include by reusing the measuredSize-box, as this leads to a subtly wrong behaviour when generating the PDF files. It can be observed by running the test suite with externalization and checking that no file is regenerated in the second

run. Reusing the box can also lead to compile errors, if the problematic graphic is the last graphic in the document.

```
472 \tikzscale@conditionalEnableExternalization{\fileName}%
473 \content%
474 }
```

`\tikzscale@resizePlotWithAxesRatioTo` `\tikzscale@resizePlotWithAxesRatioTo{<wd or ht>}{<tikzscale@width or tikzscale@height>}{<to-be-scaled content>}{<file name>}` The first argument determines if a specific width or a specific height should be achieved by resizing.

```
475 \NewDocumentCommand{\tikzscale@resizePlotWithAxesRatioTo}{mmmm}{%
476 \def\dimension{#1}%
477 \def\variable{#2}%
478 \def\content{#3}%
479 \def\fileName{#4}%
480 \gdef\tikzscale@oldSizeDifference{0pt}%
481 \tikzscale@preparePlot
```

Deactivate the externalization, as the measurements to determine the correct size should not be externalized.

```
482 \tikzscale@conditionalDisableExternalization
```

Improve the solution iteratively until it is good enough.

```
483 \foreach \l in {1,...,\maxTestIterations}{%
484 \tikzscale@measureSize{\measuredSize}{\dimension}{\content}%
```

Determine the remaining error and check if it is larger than a threshold.

```
485 \pgfmathsetmacro{\sizeDifference}{\measuredSize - \requestedSize}%
```

Output error in current iteration for debugging.

```
486 % sizeDifference: \sizeDifference\\% Debugging
```

Optimize if the absolute difference is too large.

```
487 \tikzscale@ifSizeDifference{\sizeDifference}{%
488 \tikzscale@ifIsWidth{\dimension}{%
489 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@width - \sizeDifference}%
490 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@width / \requestedAxisRatio}%
491 }{%
492 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@height - \sizeDifference}%
493 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@height * \requestedAxisRatio}%
494 }%
495 \tikzscale@ifSizeDifference{\sizeDifference-\tikzscale@oldSizeDifference}{%
496 }{%
```

Restore the externalization state in order to have strict enable-disable-call-pairing.

```
497 \tikzscale@conditionalEnableExternalization{\fileName}%
498 \tikzscale@includeNormalTikzpicture{#4}%
499 \gdef\tikzscale@alreadyIncluded{true}%
500 \breakforeach
501 }%
502 \pgfmathsetglobalmacro{\tikzscale@oldSizeDifference}{\sizeDifference}%
503 }{%
```

```

504 \breakforeach
505 }%
506 }%
507 \ifdef{\tikzscale@alreadyIncluded}{%
508 \global\undef\tikzscale@alreadyIncluded%
509 }{%

```

Do a last measurement to be able to warn if the size does not fit good enough. This measurement has to be done before possibly reactivating the externalization, as measurements with activated externalization can lead to wrong measurement results (possibly due to calling `\shipout` inside of the measurement). The assumption is, that the real graphic size does not change if the externalization gets activated, which all tests seem to confirm.

```

510 \tikzscale@measureSize{\measuredFinal}{\dimension}{\content}%
511 \tikzscale@warnIfSizeDifference{\measuredFinal}{\requestedSize}{\fileName}%

```

Finally, externalize and include the graphic with the final size. The graphic must not be include by reusing the measuredSize-box, as this leads to a subtly wrong behaviour when generating the PDF files. It can be observed by running the test suite with externalization and checking that no file is regenerated in the second run. Reusing the box can also lead to compile errors, if the problematic graphic is the last graphic in the document.

```

512 \tikzscale@conditionalEnableExternalization{\fileName}%
513 \content%
514 }%
515 }

```

```

\tikzscale@ifIsWidth \tikzscale@ifIsWidth{<condition>}{<true>}{<false>}

```

```

516 \def\tikzscale@ifIsWidth#1{%
517 \ifedefequal{#1}{\wd}%
518 }

```

```

\tikzscale@measureSize

```

```

519 \newsavebox{\tikzscale@measuredSize}
\measureSize{<result variable name>}{<\wd or \ht>}{<to-be-measured content>}
520 \def\tikzscale@measureSize#1#2#3{%
521 \sbox{\tikzscale@measuredSize}{#3}%
522 \pgfmathsetmacro{#1}{#2\tikzscale@measuredSize}%
523 }

```

```

\tikzscale@ifSizeDifference \tikzscale@ifSizeDifference{<size>}{<executed if true>}{<executed if false>}

```

```

524 \def\tikzscale@ifSizeDifference#1#2#3{%
525 \pgfmathparse{abs(#1)}%
526 \ifdimgreater{\pgfmathresult pt}{\tikzscale@accuracy}{%
527 #2%
528 }{%
529 #3%
530 }%
531 }%

```

```

\tikzscale@measuredSize \tikzscale@warnIfSizeDifference{<firstSize>}{<secondSize>}{<file name>}
532 \def\tikzscale@warnIfSizeDifference#1#2#3{%
533 \tikzscale@ifSizeDifference{#1-#2}{%
534 \PackageWarning{tikzscale}{Scaling of #3 was only\MessageBreak accurate to \pgfmathresult pt}%
535 }{}%
536 }

```

#### conditionalDisableExternalization

```

537 \NewDocumentCommand{\tikzscale@conditionalDisableExternalization}{}{}%
538 \tikzscale@ifExternalizationActive{%
539 \tikzexternaldisable
540 \def\tikzscale@externalizationWasDisabled{}%
541 }{}%

```

Restore the endlinchar here and not in the general `\tikzexternaldisable` code, as it should only be restored if `\includegraphics` had been called and not if a `tikzpicture` was called directly without using `\includegraphics`. If the externalization has not been loaded, the endlinchar would be redefined twice (which would probably also do not much harm).

```

542 \ifExternalizationLoaded{%
543 \tikzscale@addRestoreEndLineCharToTikzpicture
544 }{}%

```

The pause command defined by the Beamer class creates additional slides when called multiple times due to `tikzscale`'s scaling. Thus, deactivate it during the scaling tests, if it is defined.

```

545 \ifdef{\pause}{}%
546 \LetLtxMacro{\tikzscale@oldpause}{\pause}%
547 \RenewDocumentCommand{\pause}{o}{}%
548 }{}%
549 }

```

#### conditionalEnableExternalization

Activate externalization of TikZ graphics iff it had been active before definitely disabling it for measurement purposes. The argument contains the file name.

```

550 \NewDocumentCommand{\tikzscale@conditionalEnableExternalization}{m}{}%

```

For the externalization, set correct file name and only externalize the graphic with the final size. This produces a [known bug](#)

```

551 % \tikzsetnextfilename{#1}%

```

Get the current directory as a string and use it as an prefix, so that the graphic's PDF is generated in a subdirectory if the `tikz` file is located in a subdirectory, too. This is necessary, as the PDF file is searched for in the subdirectory in this case. This might be unnecessary due to the newly created path lookup logic.

```

552 % \expandafter\tikzsetexternalprefix\expandafter{\tikzscale@pwd}%
553 % \expandnext{\tikzsetexternalprefix}{\tikzscale@pwd}%
554 \ifdef{\tikzscale@externalizationWasDisabled}{}%
555 \tikzexternalenable
556 \undef\tikzscale@externalizationWasDisabled
557 }{}%

```

Restore the `endlinechar` here and not in the general `\tikzexternalenable` code, as it should only be restored if `\includegraphics` had been called and not if a `tikzpicture` was called directly without using `\includegraphics`. If the externalization has not been loaded, the `endlinechar` would be redefined twice (which would probably also do not much harm).

```
558 \ifExternalizationLoaded{%  
559 \tikzscale@addRestoreEndLineCharToTikzpicture  
560 }{}
```

Reactivate Beamer's `pause` command if defined.

```
561 \ifdef{\pause}{%  
562 \LetLtxMacro{\pause}{\tikzscale@oldpause}%  
563 }{}
```

```
564 }
```