

The tikzsymbols package*

Ben Vitecek
b.vitecek@gmx.at
GitHub

October 14, 2021

Abstract

Some symbols created using tikz.
For differences between the releases see section 2.
English is (still) not my native language so there (still) might be some
errors¹ ☹.

Contents

1	Introduction	2
2	Important changes	3
3	Options	3
3.1	Load-time Options	3
3.1.1	marvosym (true/false)	4
3.1.2	prefix (<i><string></i>)	4
3.2	Normal Options	5
3.2.1	draft (true/false)	5
3.2.2	final (true/false)	5
3.2.3	tree (true/false/on/off)	5
3.2.4	after-symbol (<i><string or command></i>)	6
3.2.5	global-scale (<i><number></i>) symbol-scale (<i><key-value list></i>)	6
3.2.6	append-style (<i><tikz' keyval></i>)	7
3.2.7	usebox (true/false)	7
3.2.8	baseline (true/false)	7
3.2.9	remember-picture (true/false)	7

*This document corresponds to tikzsymbols v4.12, dated 2021/10/14.

¹They are – of course – on purpose (expect for “avaiable” (sic!)).

4	Symbols	7
4.1	Cooking-symbols 🍳	8
4.2	Emoticons 😊	9
4.2.1	“Normal” Emoticons 🙄	9
4.2.2	“3D” Emoticons 🤪 🤨	11
4.3	Other Symbols 📺	12
4.4	Trees 🌳	13
5	Create your own tikzsymbol	14
5.1	tikzsymbols style	14
5.2	Symbol Definition	15
5.3	Using a box	19
5.4	Some other commands and variable(s)	20
6	FAQ (Known errors and problems)	20
6.1	How to get rid of the space after each symbol?	20
6.2	Using the symbols causes unwanted <i><problem></i> . How could I get rid of it?	21
6.3	I am getting the error-message Argument of \pgffor@next has an extra }	21
6.4	Another package I load already defines <i><symbol></i> .	21
6.5	Does this package store symbols in boxes and reuses them instead of creating a new picture every time?	21
6.6	Are the symbols created with the environment tikzpicture ?	21
7	Nobody is perfect	22
8	Danksagung	22
9	Changes	22

1 Introduction

As far as I can remember this package is the result of me writing a cooking book². Back then I wasn’t able to find the cooking symbols I wanted and using time, tikz, lot’s of magic (also known as “programming”, but only if the respective person knows what’s going on) and a documentation in bad grammar³ I somehow ended up with this package.

During time L^AT_EX3 became known to me and I started experimenting and programming in this (I would say due to its simplicity compared to L^AT_EX 2_ε far superior) language. Well, long story short: I was impressed. And so the idea of writing my package in L^AT_EX3 was born.

²Well, it’s one result, the other one is a cooking book.

³Not that it’ now any better.

I finally took my time and started rewriting my code using L^AT_EX3. This process can be summarized as: “What *does* this command?”, “Why did I define *this* command?” and more generally “*What* have I done?!” Well, let’s hope my code (and grammar) is better this time⁴.

Well ... that’s it, have fun!

2 Important changes

The package should behave the same way as the “old” L^AT_EX 2_ε release.

2018 Option `draft` and `final` are now local.

2017 Option `usebox` can be used during the document.

old The horribly named command `\tikzsymbolsaftersymbolinput` is not defined anymore by this package. Please use the new option `after-symbol`, in combination with the new command `\tikzsymbolssset`, see section 3 for more information.

very old The option `draft=absolute` is now obsolete and replaced by the much simpler option `draft=true`.

3 Options

Options can either be set as package options or using `\tikzsymbolssset`. Some options can only be set as package options, those are described in section 3.1.

It is recommended to use the option `draft=true` while working on the document.

<code>\tikzsymbolssset</code>	<code>\tikzsymbolssset {<keys = values>}</code>
-------------------------------	---

Most keys, except for the load-time options (section 3.1), can be set using this command.

3.1 Load-time Options

The following options *cannot* be set using `\tikzsymbolssset`.

⁴Looking at own risk. You have been warned.

3.1.1 marvosym (true/false)

```
marvosym = true / false
```

Please load tikzsymbols *after* marvosym.

marvosym also defines `\Smiley` and `\Coffeecup`. If you prefer those symbols (☺, ☕) over the tikzsymbols ones (☺, ☕) you can use this option. If set to true tikzsymbols cancels the definition of its `\Smiley` and `\Coffeecup`:

Without option “marvosym”: ☺ ☕	With option “marvosym”: ☺ ☕
<code>\usepackage{marvosym}</code>	<code>\usepackage{marvosym}</code>
<code>\usepackage{tikzsymbols}</code>	<code>\usepackage[marvosym]{tikzsymbols}</code>

This option raises an error if set `true` without loading package marvosym.

Can only be set as load-time option.

You may also use the option `prefix` (section 3.1.2).

3.1.2 prefix (<string>)

This option takes a string as value: `prefix=<string>` and adds this prefix to every command defined by this package. So setting `prefix=<prefix>` adds `<prefix>` to all commands of this package: `\<prefix>command`.

`<prefix>` should neither contain any special characters (e.g., ä, ü, ß, etc.) nor spaces.

By default it is empty, so no prefix is given, if this option is given without an argument `<prefix>` is set to `tikzsymbols`.

Can only be set as a load-time option.

For example:

```
\usepackage[prefix=tikzsym]{tikzsymbols}
```

defines `\Smiley` as `\tikzsymSmiley`, `\Kochtopf` as `\tikzsymKochtopf`, `\pot` as `\tikzsympot`, etc.

If you use this option or think about using this option the following command may be handy:

```
\tikzsymbolsuse \tikzsymbolsuse{<Symbolname>}
```

This command takes the name of the symbol *without* backslash and prints the symbol (or raises an error if the symbol is not defined). Using this command you don't have to worry about a `<prefix>`, just write the command name and this command adds automatically the given prefix to the command name.

Examples:

```
\tikzsymbolsuse{Smiley}[2] ☺
```

```

\tikzsymbolsuse{BasicTree}[1.2]{black}{red!50!black}{red}{leaf} 🌳
\tikzsymbolsuse{Ofen} 🚪
\tikzsymbolsuse{Fire}[-1.3] 🔥
etc.

```

3.2 Normal Options

Most of these options can be set either as a package-option or with `\tikzsymbolsset`.

3.2.1 draft (true/false)

draft `draft = $\langle true/false \rangle$`

While working on the document it is recommended to set this option to **true** because creating many symbols may takes some time to compile and by setting this option to **true** the symbols are replaced by plain vanilla rectangles (with approximately the same height and width as the symbols) which are faster to create.

You can also set this option during the document.

The old option `draft=absolute` is obsolete and should therefore not be used.

3.2.2 final (true/false)

final `final= $\langle true/false \rangle$`

This key has the opposite behavior of the option **draft**.

It is a boolean key and therefore accepts only **true** or **false** and is set to **true** by default. Setting it to **true** prints all symbols normally. Setting it to **false** prints plain vanilla draft-boxes instead which speeds up the compile-process.

3.2.3 tree (true/false/on/off)

tree `tree= $\langle true/on/false/off \rangle$`

This key accepts **true**, **false** and furthermore **on** and **off** (for historical reasons). The latter do exactly the same as the first ones.

This option has only an effect on the command `\BasicTree` and its derivatives (`\Springtree`, `\Summertree`, `\Autumntree` and `\Wintertree`) and substitutes them with tikz drawn boxes.

So while `draft=true` replaces the output of *all* commands with simple black boxes, `tree=true/on` only replaces the output of “tree”-commands with boxes.

It is recommended to use `draft=true`, but if you want you can use this option.

3.2.4 after-symbol (*(⟨string or command⟩)*)

after-symbol	after-symbol = {⟨string or command⟩}
--------------	--------------------------------------

Is more stable if set using `\tikzsymbolssset`. The value of this key is inserted after every command of this package. By default it is set to `\xspace`.

3.2.5 global-scale (*(⟨number⟩)*) symbol-scale (*(⟨key-value list⟩)*)

global-scale	global-scale = {⟨number⟩}
symbol-scale	symbol-scale = {symbol-1=number-1, symbol-2=number-2,...}

`global-scale` can be used to scale *all* commands by given *⟨number⟩*.


If only some specific symbols should be scaled, you may use the second option and specify which symbol or symbols (name of the symbol without backslash) should be scaled. Using the german name (if available) has the same effect as using the english one.

Note: You can scale the symbols in this package in three different ways: The first is to scale *all* symbols using `global-scale`, the second is scaling specific symbols using `symbol-scale` and the third is by using the optional argument provided by the symbols (which I call `local-scale`; e.g. `\Smiley[2]`).

The important thing is that those scaling methods *do not cancel* each other, but behave multiplicative.

If a local scale is given (e.g. `\Smiley[2]`) with `global-scale=3` the resulting scaling will be $3 \cdot 2 = 6$. Is furthermore this specific symbol is also scaled (e.g. by 1.1), the resulting scaling (for this symbol) will be $3 \cdot 1.1 \cdot 2 = 6.6$.

Examples: `\tikzsymbolssset{symbol-scale={ Smiley= 1.5 }}` ☺

`\tikzsymbolssset{symbol-scale={ Smiley= 5 }}` 

`\tikzsymbolssset{symbol-scale={ Smiley= 2, Schneebesen=2.1 }}` ☺🍷

`\tikzsymbolssset{global-scale=3,symbol-scale={ Smiley= 2, Schneebesen=2.1 }}`



Note: Using “eggbeater” instead of “Schneebesen” does the same thing.

3.2.6 append-style (*<tikz' keyval>*)

`append-style` `append-style = {<tikz' keyval>}`

With this option you can append `tikz' <keyval>` to `tikzsymbols` internal style.

Note: The style is called `__tikzsymbols` and while the name will probably not change, you are discouraged to use it directly unless it is *really* necessary (e.g. if I did something wrong).

3.2.7 usebox (true/false)

`usebox` `usebox = {<true/false>}`

In `tikzsymbols` all symbols are stored inside boxes (`\sbox`) and while I still have no idea what exactly happens, it shortens the compilation time of the document. By default this option is `true`.

The drawback is that \LaTeX has only a limited number of box registers. If you come across an error message regarding boxes try setting `usebox=false`.

3.2.8 baseline (true/false)

`baseline` `baseline = {<true/false>}`






This option mainly exists to let the commands of this package work inside `todonotes'` `\todo` command. If set to `true` it adds to each symbol of this package the `tikz` option `baseline=default`. If you do not want this, set this option to `false`. It is set to `true` by default.

3.2.9 remember-picture (true/false)

`remember-picture` `remember-picture = {<true/false>}`

Adds to each symbol created by this package the `tikz` option `remember picture=<true/false>`. It is not added by default.

4 Symbols

In this section the symbols are introduced. They  all  change 
automatically  with  text-size %.


















4.1 Cooking-symbols 🍳

`\Kochtopf`
`\pot`
`\Bratpfanne`
`\fryingpan`
`\Schneebeesen`
`\eggbeater`
`\Sieb`
`\sieve`
`\Purierstab`
`\blender`
`\Dreizack`
`\trident`
`\Backblech`
`\bakingplate`
`\Ofen`
`\oven`
`\Pfanne`
`\pan`
`\Herd`
`\cooker`
`\Saftpresse`
`\squeezer`
`\Schussel`
`\bowl`
`\Schaler`
`\peeler`
`\Reibe`
`\grater`
`\Flasche`
`\bottle`
`\Nudelholz`
`\rollingpin`
`\Knoblauchpresse`
`\garlicpress`

The following table shows all available cooking-symbols and their respective commands. The first column shows the command-names (german & english), the second the optional parameter(s). The optional parameter(s) are for both the german and the english commands the same.

$\langle scale \rangle$ can be a number between (not exactly) -1400 and (also not exactly) 1400 , default is 1 .

Da Umlaute nicht in Befehlsnamen vorkommen dürfen, werden die Umlaute ö, ä, ü durch o, a, u ersetzt.

German & English Commands		Optional parameter(s)	Output
<code>\Kochtopf</code>	<code>\pot</code>	$[\langle scale \rangle]$	
<code>\Bratpfanne</code>	<code>\fryingpan</code>	$[\langle scale \rangle]$	
<code>\Schneebeesen</code>	<code>\eggbeater</code>	$[\langle scale \rangle]$	
<code>\Sieb</code>	<code>\sieve</code>	$[\langle scale \rangle]$	
<code>\Purierstab</code>	<code>\blender</code>	$[\langle scale \rangle]$	
<code>\Dreizack</code>	<code>\trident</code>	$[\langle scale \rangle]$	
<code>\Backblech</code>	<code>\bakingplate</code>	$[\langle scale \rangle]$	
<code>\Ofen</code>	<code>\oven</code>	$[\langle scale \rangle]$	
<code>\Pfanne</code>	<code>\pan</code>	$[\langle scale \rangle]$	
<code>\Herd</code>	<code>\cooker</code>	$[\langle scale \rangle]$	
<code>\Saftpresse</code>	<code>\squeezer</code>	$[\langle scale \rangle]$	
<code>\Schussel</code>	<code>\bowl</code>	$[\langle scale \rangle]$	
<code>\Schaler</code>	<code>\peeler</code>	$[\langle scale \rangle]$	
<code>\Reibe</code>	<code>\grater</code>	$[\langle scale \rangle]$	
<code>\Flasche</code>	<code>\bottle</code>	$[\langle scale \rangle]$	
<code>\Nudelholz</code>	<code>\rollingpin</code>	$[\langle scale \rangle]$	
<code>\Knoblauchpresse</code>	<code>\garlicpress</code>	$[\langle scale \rangle]$	

4.2 Emoticons ☺

4.2.1 “Normal” Emoticons 🐱

\Smiley
 \Sadey
 \Neutrey
 \Changey
 \cChangey
 \Annoey
 \Laughey
 \Winkey
 \oldWinkey
 \Sey
 \Xey
 \Innocey
 \wInnocey
 \Cooley
 \Tongey
 \Nursey
 \Vomey
 \Walley
 \rWalley
 \Cat
 \SchrodingersCat
 \Ninja
 \Sleepy
 \NiceReapey

First column shows the commands, the second the (optional) parameter(s), the third the default-output (the only command with a mandatory argument is \Changey).

\Changey’s *scale* can be a number between (not exactly) -2000 and (not exactly) 2000 , default is 1.

\Changey’s *color* can be every defined color. Note: The color names shouldn’t contain special characters like ß, ä, ö, ...

\Changey’s *mood* has to be between -2 and 2 (1 equals \Smiley, -1 \Sadey and 0 \Neutrey).

\SchrodingersCat’s *case* can either be 1 (alive), 0 (unknown) or -1 (dead).

Commands	(Optional) parameter(s)	Output
\Smiley	[<i>scale</i>] [<i>color</i>]	☺
\Sadey	[<i>scale</i>] [<i>color</i>]	☹
\Neutrey	[<i>scale</i>] [<i>color</i>]	☺
\Changey	[<i>scale</i>] [<i>color</i>] { <i>mood</i> }	☺
\cChangey	[<i>scale</i>] [<i>color1</i>] [<i>color2</i>] [<i>color3</i>] { <i>mood</i> }	🟢
\Annoey	[<i>scale</i>] [<i>color</i>]	☹
\Laughey	[<i>scale</i>] [<i>color</i>] [<i>mouth color</i>]	☺
\Winkey	[<i>scale</i>] [<i>color</i>]	☺
\oldWinkey	[<i>scale</i>] [<i>color</i>]	☺
\Sey	[<i>scale</i>] [<i>color</i>]	☹
\Xey	[<i>scale</i>] [<i>color</i>]	☹
\Innocey	[<i>scale</i>] [<i>color</i>] [<i>halo color</i>]	😊
\wInnocey	[<i>scale</i>]	😊
\Cooley	[<i>scale</i>] [<i>color</i>]	😊
\Tongey	[<i>scale</i>] [<i>color</i>] [<i>tongue color</i>]	😊
\Nursey	[<i>scale</i>] [<i>color</i>] [<i>cap color</i>] [<i>cross color</i>]	😊
\Vomey	[<i>scale</i>] [<i>color</i>] [<i>vomit color</i>]	🤮
\Walley	[<i>scale</i>] [<i>color</i>] [<i>wall color</i>]	🌪
\rWalley	[<i>scale</i>] [<i>color</i>] [<i>wall color</i>]	🌪
\Cat	[<i>scale</i>]	🐱
\SchrodingersCat	[<i>scale</i>] { <i>case</i> }	🐱
\Ninja	[<i>scale</i>] [<i>color</i>] [<i>headband color</i>] [<i>eye color</i>]	🥷
\Sleepy	[<i>scale</i>] [<i>color</i>] [<i>cap color</i>] [<i>star color</i>]	😴
\Maskey	[<i>scale</i>] [<i>color</i>] [<i>mask color</i>]	😬
\NiceReapey	[<i>scale</i>]	👻

“r” for “random generated cracks”.

Examples: `\Sadey [] [red]` 🚫

`\Cooley [-3] [cyan]` 🤖

`\Vomey [1.5] [green!80!black] [olive]` 🟢👉

`\Nursey [] [yellow] [blue] [red]` 😊.

`\Ninja [1.3] [] [violet] [red]` 🟣.

`\colorbox{yellow}{\Winkey \Annoey [-1] \Neutrey}` 😊😞😊

`\textcolor{blue}{\Sey}` 😊

`\Sleepy [1] [white] [blue] [yellow!95!black]` 🛌

`\SchrodingersCat {1}` 🐈

`\SchrodingersCat {0}` 🐈

`\SchrodingersCat {-1}` 🐈

`\Changey {-2}` 😊 `\Changey {-1.367}` 😊 `\Changey {-1}` 😊 `\Changey {0}` 😊

`\Changey {1}` 😊 `\Changey {1.41}` 😊 `\Changey {2}` 😊

`\cChangey {2}` 🟢 `\cChangey {1}` 🟢 `\cChangey {0.5}` 🟢 `\cChangey {0.1}` 🟢

`\cChangey {0}` 😊 `\cChangey {-0.5}` 😊 `\cChangey {-1}` 😊 `\cChangey {-2}` 🚫

`\cChangey [] [] [blue] {-1}` 🟣 `\cChangey [] [] [blue] {0.5}` 🟢

If you intent to change the color of `\cChangey` you may define a new command so that you do not have to write those brackets each time.

4.2.2 “3D” Emoticons 🤪 🤨

`\dSmiley`
`\dSadey`
`\dNeutrey`
`\dChangey`
`\dcChangey`
`\dAnnoey`
`\dLaughy`
`\dWinkey`
`\dSey`
`\dXey`
`\dInnocey`
`\dCooley`
`\dNinja`
`\drWalley`
`\dWalley`
`\dVomey`
`\dNursey`
`\dTongey`
`\dSleepey`
`\olddWinkey`

First column shows the commands (note: the “3D” Emoticons begin with `\d...`), the second shows the (optional) parameter(s), the third shows the default-output (the only command with a mandatory argument is `\dChangey`).

`<scale>` can be a number between a small number (under -500 for sure) and a large number (over 500 for sure), default is 1 .

`<color>` can be every defined color (see examples below). Note: The color names shouldn’t contain special characters like ß, ä, ö, ...

`\Changey`’s `<mood>` has to be between -2 and 2 (1 equals `\dSmiley`, -1 `\dSadey` and 0 `\dNeutrey`).

Commands	Optional parameter(s)	Output
<code>\dSmiley</code>	<code>[<scale>] [<color>]</code>	😊
<code>\dSadey</code>	<code>[<scale>] [<color>]</code>	😓
<code>\dNeutrey</code>	<code>[<scale>] [<color>]</code>	😐
<code>\dChangey</code>	<code>[<scale>] [<color>] {<mood>}</code>	😊
<code>\dcChangey</code>	<code>[<scale>] [<color1>] [<color2>] [<color3>] {<mood>}</code>	😬
<code>\dLaughy</code>	<code>[<scale>] [<color>] [<mouth color>]</code>	😂
<code>\dAnnoey</code>	<code>[<scale>] [<color>]</code>	😒
<code>\dWinkey</code>	<code>[<scale>] [<color>]</code>	😏
<code>\olddWinkey</code>	<code>[<scale>] [<color>]</code>	😊
<code>\dSey</code>	<code>[<scale>] [<color>]</code>	😏
<code>\dXey</code>	<code>[<scale>] [<color>]</code>	😏
<code>\dInnocey</code>	<code>[<scale>] [<color>] [<halo color>]</code>	😇
<code>\dCooley</code>	<code>[<scale>] [<color>]</code>	😏
<code>\dTongey</code>	<code>[<scale>] [<color>] [<tongue color>]</code>	😏
<code>\dNursey</code>	<code>[<scale>] [<color>] [<cap color>] [<cross color>]</code>	😏
<code>\dVomey</code>	<code>[<scale>] [<color>] [<vomit color>]</code>	🤮
<code>\dWalley</code>	<code>[<scale>] [<color>] [<wall color>]</code>	😏
<code>\drWalley</code>	<code>[<scale>] [<color>] [<wall color>]</code>	😏
<code>\dNinja</code>	<code>[<scale>] [<color>] [<headband color>] [<eye color>]</code>	😏
<code>\dSleepey</code>	<code>[<scale>] [<color>] [<cap color>] [<star color>]</code>	😏

“r” for “random generated cracks”.

Examples: `\dSadey [] [red]` 🤨

`\dCooley [-3] [cyan]` 🤪

`\dVomey [1.5] [green!70!black] [olive]` 🤮

`\dNursey [] [yellow] [blue] [red]` 🤨.

`\dNinja [1.3] [] [violet] [red]` 🤪.

`\dChangey {-2}` 😓 `\dChangey {-1.367}` 😓 `\dChangey {-1}` 😓 `\dChangey {0}` 😊
`\dChangey {1}` 😊 `\dChangey {1.41}` 😊 `\dChangey {2}` 😊

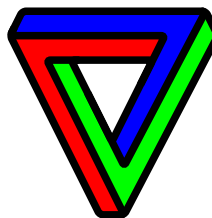
`\dcChangey{2}` 🟢 `\dcChangey{1}` 🟡 `\dcChangey{0.5}` 🟠 `\dcChangey{0.1}`
 🟡 `\dcChangey{0}` 🟠 `\dcChangey{-0.5}` 🟠 `\dcChangey{-1}` 🟠 `\dcChangey{-2}`
 🟡

`\dcChangey[] [] [blue]{-1}` 🟡 `\dcChangey[] [] [blue]{0.5}` 🟡

If you intent to change the color of `\dcChangey` you may define a new command so that you do not have to write those brackets each time.

4.3 Other Symbols 📄

<code>\Strichmaxerl</code>	<code>\Strichmaxerl</code> 's optional parameters 2–5 (<i>⟨left arm⟩</i> to <i>⟨right leg⟩</i>) can be a number between -360 and 360 (of course the number can be even greater or even smaller.). The parameters are the angles between the body and the separate parts of <code>\Strichmaxerl</code> (see examples).		
<code>\Heart</code>	<i>⟨scale⟩</i> can be a very great and a very small negative number (but I don't think, that you need so large symbols).		
<code>\dHeart</code>	<i>⟨color⟩</i> can be every defined color. Note: The color names shouldn't contain special characters like ß, ä, ö,		
<code>\Candle</code>	Commands	Optional parameter(s)	Output
<code>\Fire</code>	<code>\Strichmaxerl</code>	$[\langle scale \rangle]$ $[\langle left arm \rangle]$ $[\langle right arm \rangle]$ $[\langle left leg \rangle]$ $[\langle right leg \rangle]$	📄
<code>\Coffeecup</code>	<code>\Heart</code>	$[\langle scale \rangle]$ $[\langle color \rangle]$	❤
<code>\Chair</code>	<code>\dHeart</code>	$[\langle scale \rangle]$ $[\langle color \rangle]$	❤
<code>\Bed</code>	<code>\Candle</code>	$[\langle scale \rangle]$	🕯
<code>\Tribar</code>	<code>\Fire</code>	$[\langle scale \rangle]$	🔥
<code>\Moai</code>	<code>\Coffeecup</code>	$[\langle scale \rangle]$	☕
<code>\Snowman</code>	<code>\Chair</code>	$[\langle scale \rangle]$	🪑
	<code>\Bed</code>	$[\langle scale \rangle]$	🛏
	<code>\Moai</code>	$[\langle scale \rangle]$	🗿
	<code>\Tribar</code>	$[\langle scale \rangle]$ $[\langle color 1 \rangle]$ $[\langle color 2 \rangle]$ $[\langle color 3 \rangle]$	🔺
	<code>\Snowman</code>	$[\langle scale \rangle]$	🧊



`\Tribar[-10] [blue] [red] [green]`

`\Tribar[2.1] [blue] [blue!50] [blue!20]` 🔺

`\Strichmaxerl[1] [10] [30] [40] [4] 📄,`

`\Strichmaxerl[1.4] [210] [310] [10] [90] 📄,`

`\Strichmaxerl[2] [510] [110] [190] [990] 📄,`

```

\Strichmaxerl [0.9] [54] [28] [95] [16] %
\Strichmaxerl [] [54] [28] %
\Strichmaxerl [] [45] [45] [45] [45] %

```

```

\BasicTree
\Springtree
\Summertree
\Wintertree
\WorstTree

```

4.4 Trees 🌳






$\langle scale \rangle$ can be a number between (not exactly) -900 and (again not exactly) 900 , default is 1 .

$\langle color \rangle$ can be every defined color (see examples below). Note: The color names shouldn't contain special characters like β , \ddot{a} , \ddot{o} , \dots .

$\{\langle leaf \rangle\}$ uses the colors of $\{\langle leaf\ color\ a \rangle\}$ and $\{\langle leaf\ color\ b \rangle\}$, you can leave this one empty if you don't want leaves ($\text{\texttt{\textbackslash Wintertree}}$ is without *leaf*, see examples below).


If you are using those trees, $\text{\texttt{\textbackslash LaTeX}}$ needs longer to produce the output. So you may use the package option **tree=off**, or (better) **draft=true** (see section 3.2.1 and section 3.2.3) to make $\text{\texttt{\textbackslash LaTeX}}$ faster.

Furthermore those trees are pretty much stolen from the tikz manual.

Commands	Optional/Needed parameter(s)	Output
$\text{\texttt{\textbackslash BasicTree}}$	$[\langle scale \rangle] \{\langle trunk\ color \rangle\} \{\langle leaf\ color\ a \rangle\} \{\langle leaf\ color\ b \rangle\} \{\langle leaf \rangle\}$	see below
$\text{\texttt{\textbackslash Springtree}}$	$[\langle scale \rangle]$	
$\text{\texttt{\textbackslash Summertree}}$	$[\langle scale \rangle]$	
$\text{\texttt{\textbackslash Autumntree}}$	$[\langle scale \rangle]$	
$\text{\texttt{\textbackslash Wintertree}}$	$[\langle scale \rangle]$	
$\text{\texttt{\textbackslash WorstTree}}$	$[\langle scale \rangle]$	

$\text{\texttt{\textbackslash BasicTree}}$ examples Some “normal” trees:


```
\colorbox{green}{\BasicTree{red}{orange}{yellow}{leaf}}
```



```
\BasicTree[5]{orange!95!black}{orange!80!black}{orange!70!black}{leaf}
```



```
\BasicTree[2]{blue!65!white}{cyan!50!white}{cyan!50!white}{}
```



```
\BasicTree[-1.54]{green!20!black}{green!50!black}{green!70!black}{leaf}
```




```
\colorbox{black}{\BasicTree[3.75]{gray!80}{gray!50}{gray!40}{leaf}}
```



draftbox \BasicTree examples Some “draftbox” trees (using `tree=false`):

...and using the same trees with `tree=off/false` or `draft(=true)`:

`\colorbox{green}{\BasicTree{red}{orange}{yellow}{leaf}}` 

`\BasicTree[5]{orange!95!black}{orange!80!black}{orange!70!black}{leaf}`



`\BasicTree[2]{blue!65!white}{cyan!50!white}{cyan!50!white}{}` 

`\BasicTree[-1.54]{green!20!black}{green!50!black}{green!70!black}{leaf}`



`\colorbox{black}{\BasicTree[3.75]{gray!80}{gray!50}{gray!40}{leaf}}`



I think it's better if you define your own trees using `\newcommand` and `\BasicTree`:

```
\newcommand{\Myicetree}[1][1]{%
```

```
\BasicTree[#1]{blue!65!white}{cyan!50!white}{cyan!50!white}{}}
```

5 Create your own tikzsymbol

Suppose you have your own symbol, created in a `tikzpicture` (or something else). This package offers some commands with whom you can create your own `tikzsymbol` (including the benefits of using `symbol-scale`, `global-scale`, etc.).

5.1 tikzsymbols style

`/tikzsymbolsstyle` `/tikzsymbolsstyle`

`tikzsymbols` defines its own `tikz-style`. A public version of it is available *via* `/tikzsymbolsstyle`.

(Note: To change the internal style version use the option `append-style`. Everything added to the internal version is also added to the public version.)

5.2 Symbol Definition

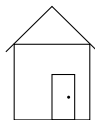
```
\tikzsymbolsdefinesymbol <symbol-name> <argument-types> <code>
```

This is the main command to define your symbol. *<symbol-name>* is the name of your symbol (without backslash) and to-be command. *<code>* is the place to put your `tikzpicture`. *<argument-types>* is somewhat similar to `xparse`'s system with some changes and only three argument types available:

- **m** is for mandatory arguments (given in curly braces).
- **B**{<default>} specifies an optional argument that inserts <default> if the optional argument (given in square-brackets) is not given *or empty*.
- **S** (for **S**cale) is the argument for scaling the symbol; the optional argument [*scale*] for each symbol is created by it. The absolute value of the scaling can be accessed by `\tikzsymbolsscaleabs`.

Example Assume you have created the following picture:

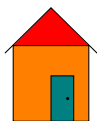
```
\begin{tikzpicture}[/\tikzsymbolsstyle]
  \fill[fill=none] (0,1) -- (0.5,1.5) -- (1,1);
  \filldraw[fill=none] (0,0) rectangle (1,1);
  \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
  \draw[fill=none] (0.5,0) rectangle (0.8,0.6);
  \fill (0.72,0.3) circle [radius=0.02];
\end{tikzpicture}%
```



Which prints:

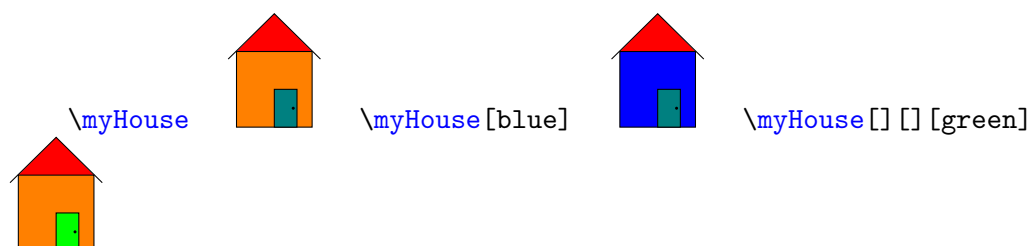
Now you want three different parts to be filled with color: The house itself, the roof and the door, let's use orange, red and teal:

```
\begin{tikzpicture}[/\tikzsymbolsstyle]
  \fill[fill=red] (0,1) -- (0.5,1.5) -- (1,1);
  \filldraw[fill=orange] (0,0) rectangle (1,1);
  \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
  \draw[fill=teal] (0.5,0) rectangle (0.8,0.6);
  \fill (0.72,0.3) circle [radius=0.02];
\end{tikzpicture}%
```




Satisfied with this you put it into `\tikzsymbolsdefinesymbol` and make it to have three optional arguments (using the B type) to change the colors. The default colors themselves are put into the argument of B:

```
\tikzsymbolsdefinesymbol {myHouse} { B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle]
    \fill[fill=#2] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#1] (0,0) rectangle (1,1);
    \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
    \draw[fill=#3] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.75,0.25) circle [radius=0.02];
  \end{tikzpicture}%
}
```



Although it looks good, you also want the ability to scale it. Furthermore, you want the symbol to be a bit smaller by default. Scaling with the text size and having a thicker line width would also be not bad. Looking at the tikz manual we can get the options necessary: `scale`, `line width`, `x` and `y`. Let's start with `x` and `y` to make the house around the same size as an uppercase letter. Let's start with `x=1ex`, `y=1ex`


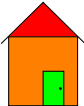
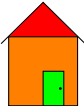
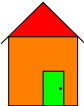
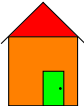
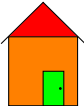
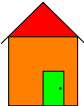
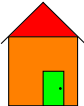
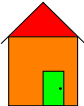
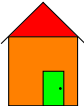
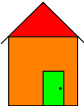
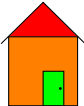

```
\tikzsymbolsdefinesymbol {myHouse} { B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle, x=1ex,y=1ex]
    \fill[fill=#2] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#1] (0,0) rectangle (1,1);
    \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
    \draw[fill=#3] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.75,0.25) circle [radius=0.02];
  \end{tikzpicture}%
}
```

A . Well a bit small maybe, but around the size of the letter “A”. Increasing `x` and `y` to `1.1ex` should be good. Now add the scaling option `S` to the command, which will be the new first argument. As it will be the new `#1`, the other arguments should be increased by 1 each.


```

\tikzsymbolsdefinesymbol {myHouse} { S B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle, x=1.1ex, y=1.1ex, scale=#1]
    \fill[fill=#3] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#2] (0,0) rectangle (1,1);
    \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
    \draw[fill=#4] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.75,0.25) circle [radius=0.02];
  \end{tikzpicture}%
}

```





Now we can scale the symbol: `\myHouse[2]`  `\myHouse[5]`  `\myHouse[10]`  `\myHouse[20]`  `\myHouse[30]`  `\myHouse[40]`  `\myHouse[50]`  `\myHouse[60]`  `\myHouse[70]`  `\myHouse[80]`  `\myHouse[90]`  `\myHouse[100]`  `\tikzsymbolsset{symbol-scale={myHouse=2}}` `\myHouse` 

Something you may not notice currently, but becomes apparent once you increase the symbol to a large scale is that the line width does not scale with the symbol. The line width can be set *via* `line width`. For the border of the emoticons I generally use a line width of `0.12ex`. Multiplying it with the scaling allows the symbol to keep the line width even at larger (or smaller) scales. Let's try it:

```

\tikzsymbolsdefinesymbol {myHouse} { S B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle, x=1.1ex, y=1.1ex,
    scale=#1, line width=0.12ex*#1]
    \fill[fill=#3] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#2] (0,0) rectangle (1,1);
    \draw (-0.11,0.9) -- (0.5,1.5) -- (1.11,0.9);
    \draw[fill=#4] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.75,0.25) circle [radius=0.02];
  \end{tikzpicture}%
}

```

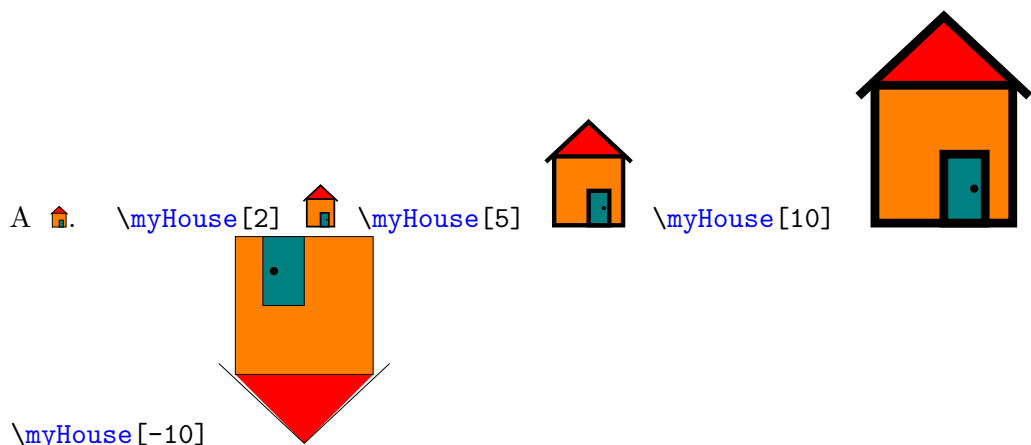
A  `\myHouse[2]`  `\myHouse[5]`  `\myHouse[10]` 

Not sure if `0.12ex` looks good for large scales. Let's use `0.07ex` and put the doorknob a bit to the left (and make it larger). Furthermore, parts of the house clip through the ceiling so let's change some coordinates there too.

```

\tikzsymbolsdefinesymbol {myHouse} { S B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle, x=1.1ex, y=1.1ex,
    scale=#1, line width=0.07ex*#1]
    \fill[fill=#3] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#2] (0,0) rectangle (1,1);
    \draw (-0.12,0.92) -- (0.5,1.5) -- (1.12,0.92);
    \draw[fill=#4] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.72,0.25) circle [radius=0.03];
  \end{tikzpicture}%
}

```

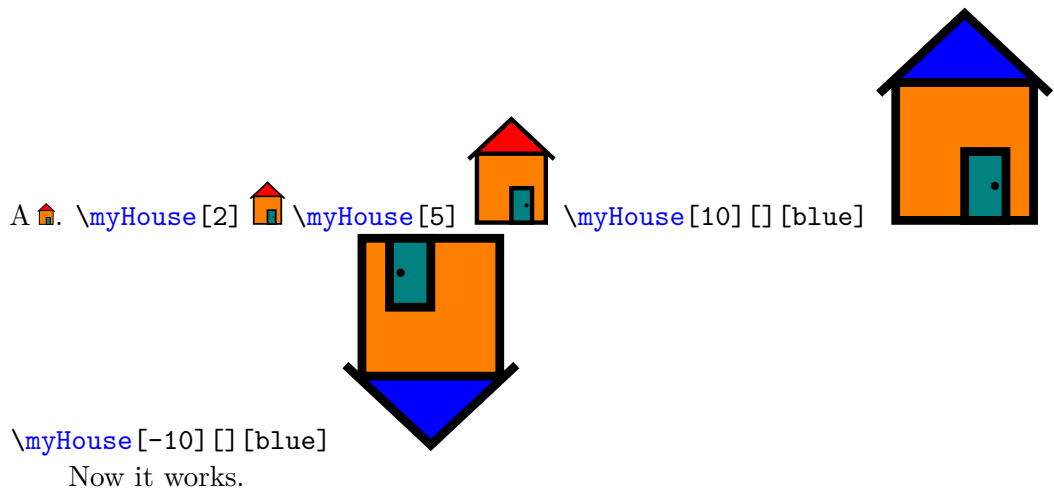


Well, looks like for negative scaling the line width does not feel so good. For this reason `\tikzsymbolsscaleabs` exists, which stores the absolute value of the scaling. Replacing `#1` with `\tikzsymbolsscaleabs` for the line widths yields:

```

\tikzsymbolsdefinesymbol {myHouse} { S B{orange} B{red} B{teal} }
{%
  \begin{tikzpicture}[/tikzsymbolsstyle, x=1.1ex, y=1.1ex, scale=#1,
    line width=0.07ex*\tikzsymbolsscaleabs]
    \fill[fill=#3] (0,1) -- (0.5,1.5) -- (1,1);
    \filldraw[fill=#2] (0,0) rectangle (1,1);
    \draw (-0.12,0.92) -- (0.5,1.5) -- (1.12,0.92);
    \draw[fill=#4] (0.5,0) -- (0.5,0.5) -- (0.8,0.5) -- (0.8,0) -- cycle;
    \fill (0.72,0.25) circle [radius=0.03];
  \end{tikzpicture}%
}

```



5.3 Using a box

If your symbol is a more complex, using it often may slow down the compilation process. In order to mitigate this we can store the symbol in a box and reuse the box instead of redrawing the symbol every time. To store the symbol, one can use the following command.


```
\tikzsymbolsprovideandusesavebox \tikzsymbolsprovideandusesavebox{<box-name>}{<box-code>}
```




The name of the box *<box-name>* (without backslash) should contain the name of the symbol and its arguments (separated by a sign). Everything inside *<box-code>* is stored in the box and repeated if used again.

Example

```
\tikzsymbolsdefinesymbol {myHouse} { S B{orange} B{red} B{teal} }
{%
  % Putting every argument in the name is important!
  \tikzsymbolsprovideandusesavebox {myHouse;#1;#2;#3;#4} {%
    \begin{tikzpicture}[/\tikzsymbolsstyle, x=1.1ex, y=1.1ex, scale=#1,
      line width=0.07ex*\tikzsymbolsscaleabs]
      \fill[fill=#3] (0,1) -- (0.5,1.5) -- (1,1);
      \filldraw[fill=#2] (0,0) rectangle (1,1);
      \draw (-0.12,0.92) -- (0.5,1.5) -- (1.12,0.92);
      \draw[fill=#4] (0.5,0) -- (0.5,0.5)
        -- (0.8,0.5) -- (0.8,0) -- cycle;
      \fill (0.72,0.25) circle [radius=0.03];
    \end{tikzpicture}%
  }%
}
```

Now the first time the symbol is used it is stored in a box. Using the symbol again in the same conditions resumes the content of the box. **Keep in mind:** It is important to give `\tikzsymbolsprovideandusesavebox` *all* arguments your symbol has. If you add or remove one option you need to update the $\langle box-name \rangle$ accordingly.



`\myHouse`  `\myHouse[10]`

`\myHouse[3] [] [blue]` 

This concludes this example.

5.4 Some other commands and variable(s)

Here some other commands which may be useful

```
\tikzsymbolssetscaleabs
\tikzsymbolsscaleabs
```

```
\tikzsymbolssetscaleabs{<dimension>}
\tikzsymbolsscaleabs
```

`\tikzsymbolssetscaleabs` sets the value of `\tikzsymbolsscaleabs` equal to the absolute value of $\langle dimension \rangle$.

Other public functions and variables which need to be used inside the `\ExplSyntaxOn` and `\ExplSyntaxOff` environment.

```
\tikzsymbols_create_draftbox:nn
\tikzsymbols_create_squared_draftbox:n
\l_tikzsymbols_if_opt_draft_bool
```

```
\tikzsymbols_create_draftbox:nn {<x-dim>} {<y-dim>}
\tikzsymbols_create_squared_draftbox:n {<dimension>}
\l_tikzsymbols_if_opt_draft_bool
```

`\tikzsymbols_create_draftbox:nn` creates draft-box (option `draft=true`) with length $\langle x-dim \rangle$ and height $\langle y-dim \rangle$.

`\tikzsymbols_create_squared_draftbox:n` does the same thing, it just takes one argument and creates a square.

`\l_tikzsymbols_if_opt_draft_bool` is a public variable that stores the value of the `draft` option.

6 FAQ (Known errors and problems)

Or “Questions I assume would be frequently asked, if people would frequently ask questions”.

6.1 How to get rid of the space after each symbol?

By default the package adds `\xspace` after each command. To remove it use the option `after-symbol`. Using

```
\tikzsymbolsset{after-symbol={}}
```

removes the `\xspace` command and thus the unwanted space.

6.2 Using the symbols causes unwanted *<problem>*. How could I get rid of it?

This could have something to do with question 6.5 (after you made sure that the symbols cause the problem). Try using setting the option `usebox=false` and recompile a few times. If the problem persists, please send a bug report (section 7).

6.3 I am getting the error-message Argument of \pgffor@next has an extra }

If you encounter an error message like

```
Argument of \pgffor@next has an extra }
```

while using `babel` with e.g. language “français” and for example `\Coolley` you may add

```
\usetikzlibrary{babel}
```

to your preamble. This should (hopefully) fix the problem.

6.4 Another package I load already defines *<symbol>*.

You can override pretty much every symbol simply by loading `tikzsymbols` last as it defines the symbols via `\DeclareDocumentCommand` (see `xparse`).

If you want to use the symbols of both packages you may have a look at option `prefix`.

6.5 Does this package store symbols in boxes and reuses them instead of creating a new picture every time?

Yes, it does. It can become a problem if \LaTeX runs out of boxes. If this happens, use `usebox=false`.

Furthermore, `tikz` allows to reference pictures using e.g. `remember picture`. This also influences the symbols of `tikzsymbols`. As those symbols are stored and copied for printing, labels attached to the symbols get repeated. In this case, also try using `usebox=false` (or try the option `remember-picture=false`).

6.6 Are the symbols created with the environment `tikzpicture`?

Yes, they are.

7 Nobody is perfect

If you find a bug please send me a mail (or report it on GitHub) involving a *minimal example* showing the bug and a short description (english or german). Please mention (if you are writing a mail) “tikzsymbols” in the header, “gmx” has a habit of putting mails into the spam-folder and it helps me to recognize those mails faster. This can also be the reason why I may need some time to answer the mail.

As I am also new to GitHub, I also may take longer to answer, at least until I figured out how to get a mail if a new issue is created.

Suggestions are also welcome.

8 Danksagung

I would like to thank all users for providing bug reports and helping to improve this package.

Furthermore many thanks to my brother helping me improving the symbols.

9 Changes

See the “README.md” file.