MENUKEYS

Tobias Weh
mail@tobias-weh.de
http://www.tobias-weh.de
http://www.ctan.org/pkg/menukeys
macros▸latex▸contrib▸menukeys

2012/04/11 — v1.1b

Abstract

This package is build to format menu sequences, paths and keystrokes.

You're welcome to send me feedback, questions, bug reports and feature request. If you like to support this package – especially improving or proof-reading the manual – send me an e-mail, please.

Many thanks to Ahmed Musa, who provided the list parsing code at http://tex.stackexchange.com/a/44989/4918.

Contents

1 Introduction

The `menukeys` package is mainly designed to parse and print sequences of software menus, folders and files or keystrokes. The most predefined styles use the power of Ti*k*Z[1] to format the output

For example if you want to tell the reader of a manual how to set the ruler unit you may type

```
To set the unit of the rulers go to \menu{Extras > Settings > Rulers}
and choose between millimetres, inches and pixels. The short cut
to view the rulers is \keys{cmd + R}. Pressing these keys again
will hide the rulers.

The standard path for saving your document is \directory{Macintosh HD/Users/
Your Name/Documents} but you can change it at \menu{Extras > Settings
> Saving} by clicking \menu{Change save path}.
```

and get this:

To set the unit of the rulers go to Extras › Settings › Rulers and choose between millimetres, inches and pixels. The short cut to view the rulers is cmd + R. Pressing these keys again will hide the rulers.

The standard path for saving your document is Macintosh HD ▸ Users ▸ Your Name ▸ Documents but you can change it at Extras › Settings › Saving by clicking Change save path.

The package is loaded as usual via

```
\usepackage{menukeys}
```

2 Installation

To install `menukeys` manually run

```
latex menukeys.ins
```

and copy `menukeys.sty` to a path where LaTeX can find it.

To typeset this manual run

```
pdflatex menukeys.dtx
makeindex -s gglo.ist -o menukeys.gls menukeys.glo
makeindex -s gind.ist -o menukeys.ind menukeys.idx
pdflatex menukeys.dtx
pdflatex menukeys.dtx
```

[1] See http://www.ctan.org/pkg/pgf.

3 Package options

These are the possible options:

definemenumacros (opt.)

definemenumacros: Most of `menukeys`' macros should not conflict with other packages[2] but the predefined menu macros should be short and easy to read commands, which means that `\menu{A,B,C}` is preferred against `\printmenusequence{A,B,C}`. For that it's not unlikely that these conflict with other packages. To prevent this you cant tell `menukeys` to not define them by calling the option `definemenumacros=false`. The default value is `true`.

If you do so you have to define your own menu macros, see section ?? for details.

definekeys (opt.)

definekeys: Equal to `definemenumacros` for the key macros. The default value is `true`.

mackeys (opt.)

mackeys: This option allows you to decide wether the mac keys are shown as text (`mackeys=text`) or symbols (`mackeys=symbols`). The default value is `symbols`

os (opt.)

os: You can specify the OS by saying `os=mac` or `os=win`. This will cause some key macros to be rendered different. The default value is `mac`.

4 Usage

4.1 Basics

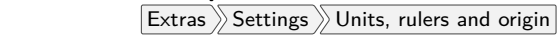
\menu
\directory
\keys

`menukeys` comes with three "menu macros" that parse and print lists. We have `\menu{`⟨*menu sequence*⟩`}`, with `>` as default input list separator, `\directory{`⟨*path and files*⟩`}` with `/` as default separator and `\keys{`⟨*keystrokes*⟩`}` with `+` as default separator. You've seen examples for all of them in section ??.

These macros have also an optional argument to set the input list separator. E.g. if you want to put in your menus with `,` instead of `>` you can say `\menu[,]{`⟨*menu sequence*⟩`}`.[3]

The possible input separators are `/`, `=`, `*`, `+`, `,`, `;`, `:`, `-`, `>`, `<` and `bslash` (to use `\` as separator). You can hide a separator from the parser by putting a part of the sequence in braces. Spaces around the seperator will be ignored, i.e. `\keys{\ctrl+C}` equals `\keys{\ctrl + C}`.

Example `\menu[,]{Extras,Settings,{Units, rulers and origin}}` gives



[2] If you find a conflict send an e-mail.

[3] If you want to change the input separator globally it's recommended to renew the menu macro as described in section ??.

4.2 Styles

`menukeys` defines several "styles" that determine the output format of a menu macro. There are some predefined styles and others can be created by the user.

4.2.1 Predefined styles

Name: `menus`

File » Extras » Preferences Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way File » Extras » Preferences the style which name is `menus` prints the list.

Name: `roundedmenus`

File » Extras » Preferences Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way File » Extras » Preferences the style which name is `roundedmenus` prints the list.

Name: `angularmenus`

File » Extras » Preferences Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way File » Extras » Preferences the style which name is `angularmenus` prints the list.

Name: `roundedkeys`

Ctrl + Alt + Q Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way Ctrl + Alt + Q the style which name is `roundedkeys` prints the list.

Name: `shadowedroundedkeys`

Ctrl + Alt + Q Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way Ctrl + Alt + Q the style which name is `shadowedroundedkeys` prints the list.

Name: `angularkeys`

Ctrl + Alt + Q Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way Ctrl + Alt + Q the style which name is `angularkeys` prints the list.

Name: `shadowedangularkeys`

Ctrl + Alt + Q Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way Ctrl + Alt + Q the style which name is `shadowedangularkeys` prints the list.

Name: `typewriterkeys`

Alt + Q

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way Alt + Q the style which name is `typewriterkeys` prints the list.

Name: `paths`

C: ▸ User ▸ Folder ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ User ▸ Folder ▸ MyFile.tex the style which name is `paths` prints the list.

Name: `pathswithfolder`

C: ▸ User ▸ Folder ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ User ▸ Folder ▸ MyFile.tex the style which name is `pathswithfolder` prints the list.

Name: `pathswithblackfolder`

C: ▸ User ▸ Folder ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ User ▸ Folder ▸ MyFile.tex the style which name is `pathswithblackfolder` prints the list.

The following three styles allow paths elements to be hyphened, but they insert only a line break without a hyphen dash. Note that they only work with `T1` and `OT1` encoding, at least I tested only these ones.

Name: `hyphenatepaths`

C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex the style which name is `hyphenatepaths` prints the list.

Name: `hyphenatepathswithfolder`

C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex the style which name is `hyphenatepathswithfolder` prints the list.

Name: `hyphenatepathswithblackfolder`

C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex Single

This is some more or less blind text, to demonstrate how the sequence looks in text. This is way C: ▸ Database ▸ User ▸ ALongUserNameHere ▸ ALongerFolderNameAtThisPlace ▸ MyFile.tex the style which name is `hyphenatepathswithblackfolder` prints the list.

`\drawtikzfolder` **Hint** The folder is drawn with the command `\drawtikzfolder[`⟨*frontcolor*⟩`]` which is part of `menukeys`.

4.2.2 Declaring styles

`\newmenustylesimple` The simplest way to define a new style is to use `\newmenustylesimple`. It has six arguments: `\newmenustylesimple`⟨`*`⟩`{`⟨*name*⟩`}[`⟨*pre*⟩`]{`⟨*style*⟩`}[`⟨*sep*⟩`][`⟨*post*⟩`]{`⟨*theme*⟩`}`

name is the name of the new style. It must follow the specifications of TEX control sequences, which means it must contain only letters and no numbers.

pre is the code which is executed before a menu macro.

style is the style for the first list element. It has to be a Ti*k*Z-style which is applied to a `node`, e.g. `draw,blue`.

sep is the code executed between the lists elements, e.g. some space or a symbol.

post is the code which is executed after a menu macro.

theme is a color theme (see section **??**).

Example Let us consider we want a list that prints a frame around it's elements and seperates them by a star. We can use

```
\newmenustylesimple{mystyle}{draw}[$\ast$]{mycolors}
```

`\newmenustyle` The more advanced command is `\newmenustyle`. It has nine arguments: `\newmenustyle`⟨`*`⟩`{`⟨*name*⟩`}[`⟨*pre*⟩`]{`⟨*first*⟩`} [`⟨*sep*⟩`]{`⟨*mid*⟩`}{`⟨*last*⟩`}{`⟨*single*⟩`} [`⟨*post*⟩`]{`⟨*theme*⟩`}`

name is the name of the new style. It must follow the specifications of TEX control sequences, which means it must contain only letters and no numbers.

pre is the code which is executed before a menu macro.

first is the style for the first list element. It has to be a Ti*k*Z-style which is applied to a `node`, e.g. `draw,blue`.

sep is the code executed between the lists elements, e.g. some space or a symbol.

mid is the style for all elements between the first and the last one. It has to be a Ti*k*Z-sytle.

last is the style for the last list element. It has to be a Ti*k*Z-sytle.

single this style is used if the list contains only one element. It has to be a Ti*k*Z-sytle.

post is the code which is executed after a menu macro.

theme is a color theme (see section **??**).

Example We can extend the previous example and desire that the first and the last element became red, and a single element should have a dashed frame. Furthermore the menu sequence should be preceded and followed by a bullet point:

```
\newmenustyle{mystyle}[$\bullet$]{draw,red}[$\ast$]%
   {draw}{draw,red}{draw,dashed}[$\bullet$]
```

If the Ti*k*Z `node` system doesn't fit you needs there are the **starred versions**: Use them and the arguments ⟨*first*⟩, ⟨*mid*⟩, ⟨*last*⟩, ⟨*single*⟩ can be any LATEX code.

`\CurrentMenuElement` To access the current list element use `\CurrentMenuElement`.

Example consider that we want all menu elements simple be fat and not drawn with a Ti*k*Z `node`. The separator should be the star again:

```
\newmenustylesimple*{mystyle}{\textbf{\CurrentMenuElement}}[$\ast$]
```

`\usemenucolor` If you want to make your own style you must take care of using the color theme. To access a color of the currently applied theme while defining a style use `\usemenucolor{`⟨*element*⟩`}` (See section **??** for details about possible elements).

4.2.3 Copying styles

`\copymenustyle` To copy an existing style to a new style us `\copymenustyle {`⟨*copy*⟩`}{`⟨*original*⟩`}`.

Example To copy the definition of `mystyle` to `mycopy` use

```
\copymenustyle{mycopy}{mystyle}
```

4.2.4 Changing styles

`\changemenuelement` The simplest change we can imagine is to change a single element or the color theme of an existing style. For the first case there is `\changemenuelement`⟨`*`⟩`{`⟨*name*⟩`}{`⟨*element*⟩`}{`⟨*definition*⟩`}`, where the starred version works like the one of `\newmenustyle` does.

Example to change the single element of `mystyle` frome dashed to solid use the following code. You may save the original style by copying it as described above.

```
\changemenuelement{mystyle}{single}{draw}
```

`\changemenucolortheme` To satisfy the second case use `\changemenucolortheme {`⟨*name*⟩`}{`⟨*color theme*⟩`}`.

Example To change the color theme of `mystyle` to `myothercolors` call

```
\changemenucolortheme{mystyle}{myothercolors}
```

`\renewmenustylesimple` `\providemenustylesimple` `\renewmenustyle` `\providemenustyle` The next level is redefing a style. This package provides the following macros the work like their LaTeX-paragons and have the same arguments as the above described macros: `\renewmenustylesimple`, `\providemenustylesimple`, `\renewmenustyle` and `\providemenustyle`.

4.3 Color themes

To make the colors of a style became changeable without touching the style itself, `menukeys` uses "color themes". Every color theme must contain three color definitions that can be used to draw a `node` background, a `node` frame and a text color.

4.3.1 Predefined themes

There are two predefined color themes

Name: `gray` Background Border ■ Text ■

Name: `blacknwhite` Background Border ■ Text ■

4.3.2 Create a theme

\newmenucolortheme To create a new theme use `\newmenucolortheme`. It uses the following five arguments: `\newmenucolortheme{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*bg*⟩`}{`⟨*br*⟩`}{`⟨*txt*⟩`}`

name is the name of the theme and must contain only letters.

model is the `xcolor` color model which is used to define a color, e.g. `named`, `rgb`, `cmyk`, ...

bg is the color definition for the `node` background.

br is the color definition for the `node` border.

txt is the color definition for the `node`'s text.

Example To create a theme called `mycolors` we can say

```
\newmenucolortheme{mycolors}{named}{red}{green}{blue}
```

4.3.3 Copy a theme

\copymenucolortheme To copy the definitions of one theme to another, use `\copymenucolortheme {`⟨*copy*⟩`}{`⟨*original*⟩`}`.

Example To copy the colors of `mycolors` to `copycolors` type

```
\copymenucolortheme{copycolors}{mycolors}
```

4.3.4 Change a theme

\changemenucolor If you want to change the color of a theme's element use `\changemenucolor{`⟨*name*⟩`} {`⟨*element*⟩`}{`⟨*model*⟩`}{`⟨*color definition*⟩`}`, where name is the theme's name and ⟨*element*⟩ is `bg`, `br`, or `txt`.

Example Let's change the text color of `mycolors`:

```
\changemenucolor{mycolors}{txt}{named}{gray}
```

\renewmenucolortheme To redefine a complete theme use `\renewmenucolortheme`. It works with the same arguments as `\newmenucolortheme`.

4.4 Menu macros

The "menu marcos" take a list seperated by a special symbol to print it with a menu style.

4.4.1 Predefined menu macros

See section **??**.

4.4.2 Defining or changing menu macros

\newmenumacro To define a new menu macro call `\newmenumacro{`⟨*macro*⟩`} [`⟨*input sep*⟩`]{`⟨*style*⟩`}`.

name is a LaTeX control sequence name.

input sep is the default seperator used in the input list (see section **??** for a list of valid separators).

If you don't give it the package's default (`,`) is used.

style is a menu style.

This wil give you a macro like `\`⟨*macro*⟩`[`⟨*input sep*⟩`]{`⟨*list*⟩`}`

Example Assuming you need a command to format Windows paths, you can define it with

```
\newmenumacro{\winpath}[bslash]{mystyle}
```

and then use it as e.g. `\winpath{C:\System\Deep\Deeper\YourFile.txt}`. Note that `mystyle` must be defined before you call `\newmenumacro`.

\providemenumacro \renewmenumacro There are also the two commands `\providemenumacro` and `\renewmenumacro` which take the same arguments as `\newmenumacro` and word like their LaTeX analogons.

Example To change the default input seperator of `\menu` you must know the default style (which is `menus`) and then you can say

```
\renewmenumacro{\menu}[,]{menus}
```

4.5 Keys

\shift \capslock \tab \esc \ctrl \alt \AltGr \cmd \Space \return \enter \winmenu \backspace \del \arrowkeyup \arrowkeydown \arrowkeyleft \arrowkeyright The `menukeys` package comes with some macros to print special keys in the sequences set with `\keys`. Depending on the given OS (see Section **??**) some macros behave differently to be able to use a key even if it's undefined via the `os` option macros like `\`⟨*key*⟩`mac` and `\`⟨*key*⟩`win` that will always give the right symbol.

Here is a full list of available macros:

Macro	Mac	Win.
\shift	⇧	⇧
\capslock	⇪	⇩
\tab	⟶	⭾
\esc	esc / ⎋	Esc
\ctrl	ctrl	Ctrl
\Alt	alt / ⌥	Alt
\AltGr		Alt Gr
\cmd	cmd / ⌘	
\Space		
\return	⏎	↵
\enter	⌅	Enter

Macro	Mac	Win.
\winmenu		☰
\backspace	⟵	⟵
\del	Del. / ⌦	Del.
\arrowkey{^}	↑	↑
\arrowkeyup	↑	↑
\arrowkey{v}	↓	↓
\arrowkeydown	↓	↓
\arrowkey{>}	→	→
\arrowkeyright	→	→
\arrowkey{<}	←	←
\arrowkeyleft	←	←

\arrowkey The macro \arrowkey{⟨*direction*⟩} is a little special since it takes the direction as a singe character ^, v (lower case v), > or <.

\ctrlname \delname The texts for \ctrl and \del are saved in \ctrlname and \delname respectively. So you can change them with \renewcommand.

mackeys (opt.) The rendering of some Mac macros depend on the option mackeys The different versions are shown in the table above (left: text, right: symbols).

I apologise that there are no commands for the windows key and the apple logo, but that would be a copyright infringement.

5 Known issues and bugs

- If you use the inputenc package menukeys must be loaded after it. Otherwise some key macros get corrupted.
- menukeys must be loaded after xcolor, if you load the latter with options. Otherwise you'll get an option clash Since menukeys loads xcolor internally you may pass options as global options via \documentclass.

  Example Set xcolor to cmyk model:

  ```
  \documentclass[cmyk]{article}
  \usepackage{menukeys}
  \begin{document}
     Hello World!
  \end{document}
  ```

- Since menukeys uses catoptions it may causes some problems with orther packages becaus of catoptions option handling. I recommend to load menukeys as a later package in your preamble.

If you find something to add to this list please send me an e-mail.

6 Implementation

6.1 Required packages

Load the required packages

```
\RequirePackage{xparse}
\RequirePackage{xstring}
\RequirePackage{etoolbox}
```

Furthermore we need Ti*k*Z and some of it's libraries,

```
\RequirePackage{tikz}
   \usetikzlibrary{calc,shapes.symbols,shadows}
```

the color package `xcolor` and `adjustbox` for the `typewriterkeys` sytle.

```
\RequirePackage{xcolor}
\RequirePackage{adjustbox}
```

Load `relsize` to be able to change the font size relative to the surrounding text.

```
\RequirePackage{relsize}
```

To define the list parsing commands and allow `\` as a seperator we load `catoptions`

```
\RequirePackage{catoptions}[2011/12/07]
```

6.2 Helper macros

`\tw@mk@error` `\tw@mk@warning` `\tw@mk@warning@noline`

Define macros to call `\PackageError` and warnings

```
\newcommand*{\tw@mk@error}[2][Please consult the manual for more information.]{%
   \PackageError{menukeys}{#2}{#1}%
}
\newcommand*{\tw@mk@warning}[1]{%
   \PackageWarning{menukeys}{#1}%
}
\newcommand*{\tw@mk@warning@noline}[1]{%
   \PackageWarningNoLine{menukeys}{#1}%
}
```

`\tw@mk@tempa` `\tw@mk@tempb`

Some commads for temporary use:

```
\def\tw@mk@tempa{}
\def\tw@mk@tempb{}
```

`\tw@mk@gobble@args`

Define a command to gobble arguments.

```
\DeclareDocumentCommand{\tw@mk@gobble@args}{m}{%
   \RenewDocumentCommand{\tw@mk@tempa}{#1}{}%
   \tw@mk@tempa%
}
```

6.3 Options

First we declare and process the package options

```
\RequirePackage{kvoptions}
\SetupKeyvalOptions{
  family=tw@mk,
  prefix=tw@mk@
}
\DeclareBoolOption[true]{definemenumacros}
\DeclareBoolOption[true]{definekeys}
\DeclareStringOption[mac]{os}
\DeclareStringOption[symbols]{mackeys}
\ProcessKeyvalOptions{tw@mk}\relax
```

Now we have to do some error treatment:

```
\IfSubStr{.mac.win.}{.\tw@mk@os.}{}{%
   \tw@mk@error{Unknown value for option 'os'\MessageBreak
   Possible values are 'mac' or 'win'.}%
}
\IfSubStr{.symbols.text.}{.\tw@mk@mackeys.}{}{%
   \tw@mk@error{Unknown value for option 'mackeys'\MessageBreak
   Possible values are 'symbols' or 'text'.}%
}
```

6.4 Color themes

6.4.1 Internal commands

`\tw@make@color@theme` First we define an internal command to make a color theme

```
\newcommand*{\tw@make@color@theme}[5]{%
   \definecolor{tw@color@theme@#1@bg}{#2}{#3}%
   \definecolor{tw@color@theme@#1@br}{#2}{#4}%
   \definecolor{tw@color@theme@#1@txt}{#2}{#5}%
}
```

6.4.2 User-level commands

`\newmenucolortheme` After that we define the user-level commands:

`\newmenucolortheme`

```
\newcommand*{\newmenucolortheme}[5]{%
   \@ifundefinedcolor{tw@color@theme@#1@bg}{%
      \tw@make@color@theme{#1}{#2}{#3}{#4}{#5}
   }{%
      \tw@mk@error{Color theme '#1' already defined!\MessageBreak
      Use \string\renewmenucolortheme\space instead.}
   }
}
\let\renewmenucolortheme\tw@make@color@theme
```

`\changemenucolor` Lastyle we define the changing and copying commands

`\copymenucolortheme`

```
\newcommand*{\changemenucolor}[4]{%
```

```
   \IfSubStr{ bg br txt }{ #2 }{%
      \definecolor{tw@color@theme@#1@#2}{#3}{#4}%
   }{%
      \tw@mk@error{No such color element ('#2')!\MessageBreak
      Possible values are bg, br and txt.}
   }%
}
\newcommand*{\copymenucolortheme}[2]{%
   \@ifundefinedcolor{tw@color@theme@#1@bg}{%
      \colorlet{tw@color@theme@#1@bg}{tw@color@theme@#2@bg}%
      \colorlet{tw@color@theme@#1@br}{tw@color@theme@#2@br}%
      \colorlet{tw@color@theme@#1@txt}{tw@color@theme@#2@txt}%
   }{%
      \tw@mk@error{Color theme '#1' already defined!\MessageBreak
      Use \string\renewmenucolortheme\space instead.}
   }
}
```

\changemenucolortheme To be able to change the color theme of a style we must define this:

```
\newcommand{\changemenucolortheme}[2]{%
   \ifcsundef{tw@style@#1@pre}{%
      \tw@mk@error{Style '#1' undefined!\MessageBreak
      Maybe you misspelled it?}%
   }{%
      \@ifundefinedcolor{tw@color@theme@#2@bg}{%
         \tw@mk@error{Color theme '#2' is not defined!}%
      }{%
         \csdef{tw@style@#1@color@theme}{#2}%
      }%
   }%
}
```

\usemenucolor To use a color of a theme we define `\usemenucolor` as following.

```
\newcommand{\usemenucolor}[1]{%
   tw@color@theme@\tw@current@color@theme @#1%
}
```

6.4.3 Predefined themes

There are two predefined color themes

```
\tw@make@color@theme{gray}{gray}{0.95}{0.3}{0}
\tw@make@color@theme{blacknwhite}{gray}{1}{0}{0}
```

6.5 Styles

The style generating commands will set some commands that are named like `\tw@style@`⟨*name*⟩`@`⟨*element*⟩.

\tw@default@sep
\tw@default@pre
\tw@default@post

Before we can define the internal declaring macro to use it later in the the user level commands, we have to set some defaults for the optional arguments

```
\newcommand{\tw@default@sep}{%
   \hspace{0.2em plus 0.1em minus 0.5em}%
}
\newcommand{\tw@default@pre}{}
\newcommand{\tw@default@post}{}
```

6.5.1 Internal commands

Now we can define the internal commands.

\tw@declare@style@simple

Our first step is to define the simple command.

```
\DeclareDocumentCommand{\tw@declare@style@simple}{%
   s m O{\tw@default@pre} m O{\tw@default@sep} O{\tw@default@post} m
}{%
   \csdef{tw@style@#2@color@theme}{#7}%
   \csdef{tw@style@#2@pre}{#3}%
   \csdef{tw@style@#2@sep}{#5}%
   \csdef{tw@style@#2@post}{#6}%
   \IfBooleanTF{#1}{%
      \csdef{tw@style@#2@single}{#4}%
      \csdef{tw@style@#2@first}{#4}%
      \csdef{tw@style@#2@mid}{#4}%
      \csdef{tw@style@#2@last}{#4}%
   }{%
      \csdef{tw@style@#2@single}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@first}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@mid}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@last}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
   }%
}
```

\tw@declare@sytle
\tw@declare@sytle@extra@args

The next step is to create the extended command. This command must have ten arguments (including the star) so we have to define a helping macro to grab the last two macros.

```
\DeclareDocumentCommand{\tw@declare@sytle@extra@args}{%
   O{\tw@default@post} m
}{%
   \csdef{tw@style@\tw@current@style @post}{#1}%
   \csdef{tw@style@\tw@current@style @color@theme}{#2}%
```

```
}
```

Now we can define `\tw@declare@style`:

```
\DeclareDocumentCommand{\tw@declare@style}{%
   s m O{\tw@default@pre} m O{\tw@default@sep} m m m
}{%
   \def\tw@current@style{#2}
   \csdef{tw@style@#2@pre}{#3}%
   \csdef{tw@style@#2@sep}{#5}%
   \IfBooleanTF{#1}{%
      \csdef{tw@style@#2@single}{#8}%
      \csdef{tw@style@#2@first}{#4}%
      \csdef{tw@style@#2@mid}{#6}%
      \csdef{tw@style@#2@last}{#7}%
   }{%
      \csdef{tw@style@#2@single}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#8]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@first}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@mid}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#6]{\strut\CurrentMenuElement};}%
      \csdef{tw@style@#2@last}{%
         \tikz[baseline=(tw@node.base)]%
         \node(tw@node)[#7]{\strut\CurrentMenuElement};}%
   }%
   \tw@declare@sytle@extra@args%
}
```

6.5.2 User-level commads

newmenustylesimple
renewmenustylesimple
providemenustylesimple
newmenustyle
renewmenustyle
providemenustyle

It's time to define the user-level commands now:

```
\NewDocumentCommand{\newmenustylesimple}{s m}{%
   \ifcsundef{tw@style@#2@pre}{%
      \IfBooleanTF{#1}{%
         \tw@declare@style@simple*{#2}%
      }{%
         \tw@declare@style@simple{#2}%
      }%
   }{%
      \tw@mk@error{Style '#2' already defined!\MessageBreak
      Use \string\renewmenustylesimple\space instead.}%
      \tw@mk@gobble@args{o m o o m}%
   }%
}
\NewDocumentCommand{\renewmenustylesimple}{s m}{%
   \IfBooleanTF{#1}{%
      \tw@declare@style@simple*{#2}%
```

```
   }{%
     \tw@declare@style@simple{#2}%
   }%
}
\NewDocumentCommand{\providemenustylesimple}{s m}{%
   \ifcsundef{tw@style@#2@pre}{%
      \IfBooleanTF{#1}{%
         \tw@declare@style@simple*{#2}%
      }{%
         \tw@declare@style@simple{#2}%
      }%
   }{%
      \tw@mk@warning{Trying to provide style '#2' failed,\MessageBreak
      because it's already defined.\MessageBreak
      You may use \string\renewmenustylesimple\space instead.}%
      \tw@mk@gobble@args{o m o o m}%
   }%
}

\NewDocumentCommand{\newmenustyle}{s m}{%
   \ifcsundef{tw@style@#2@pre}{%
      \IfBooleanTF{#1}{%
         \tw@declare@style*{#2}%
      }{%
         \tw@declare@style{#2}%
      }%
   }{%
      \tw@mk@error{Style '#2' already defined!\MessageBreak
      Use \string\renewmenustyle\space instead.}%
      \tw@mk@gobble@args{o m o m m m o m}%
   }%
}
\NewDocumentCommand{\renewmenustyle}{s m}{%
   \IfBooleanTF{#1}{%
      \tw@declare@style*{#2}%
   }{%
     \tw@declare@style{#2}%
   }%
}
\NewDocumentCommand{\providemenustyle}{s m}{%
   \ifcsundef{tw@style@#2@pre}{%
      \IfBooleanTF{#1}{%
         \tw@declare@style*{#2}%
      }{%
         \tw@declare@style{#2}%
      }%
   }{%
      \tw@mk@warning{Trying to provide style #2 failed,\MessageBreak
      because it's already defined.\MessageBreak
      You may use \string\renewmenustyle\space instead.}%
```

```
      \tw@mk@gobble@args{o m o m m m o m}%
   }%
}
```

6.5.3 Copying and changing

\copymenustyle The last two steps in this part are to define a comand to copy styles

```
\newcommand*{\copymenustyle}[2]{%
   \ifcsundef{tw@style@#1@pre}{%
      \ifcsundef{tw@style@#2@pre}{%
         \tw@mk@error{Can't copy not existing style ('#2')!}%
      }{%
         \csletcs{tw@style@#1@pre}{tw@style@#2@pre}%
         \csletcs{tw@style@#1@post}{tw@style@#2@post}%
         \csletcs{tw@style@#1@sep}{tw@style@#2@sep}%
         \csletcs{tw@style@#1@single}{tw@style@#2@single}%
         \csletcs{tw@style@#1@first}{tw@style@#2@first}%
         \csletcs{tw@style@#1@mid}{tw@style@#2@mid}%
         \csletcs{tw@style@#1@last}{tw@style@#2@last}%
         \csletcs{tw@style@#1@color@theme}{tw@style@#2@color@theme}
      }%
   }{%
      \tw@mk@error{Style '#1' already exists!}%
   }%
}
```

\changemenuelement and one to change a single element of a style.

```
\NewDocumentCommand{\changemenuelement}{s m m m}{%
   \ifcsundef{tw@style@#2@pre}{%
      \tw@mk@error{Style '#2' undefined.}%
   }{%
      \IfSubStr{ single first middle last pre post sep }{ #3 }{%
         \IfBooleanTF{#1}{%
            \csdef{tw@style@#2@#3}{#4}%
         }{%
            \IfSubStr{ pre post sep }{ #3 }{%
               \csdef{tw@style@#2@#3}{#4}%
            }{%
            \csdef{tw@style@#2@#3}{%
               \tikz[baseline=(tw@node.base)]%
               \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
            }%
         }%
      }{\tw@mk@error{No element '#3'. Possible values are\MessageBreak
        single, first, middle, last, pre, post or sep.}}%
   }%
}
```

6.5.4 Predefined styles

We define several styles for menu sequences, paths and keystrokes.

`tw@set@tikz@colors` First we define a Ti*k*Z-style to apply the color theme to a node easily

```
\tikzstyle{tw@set@tikz@colors}=[%
   draw=\usemenucolor{br},
   fill=\usemenucolor{bg},
   text=\usemenucolor{txt},
]
```

Now we can define the styles. To keep the most settings of a style together we make additional Ti*k*Z-styles instead of setting everything directly to the `node`s.

```
\tikzstyle{tw@menus@base}=[%
   tw@set@tikz@colors,
   rounded corners=0.15ex,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
   signal,
   signal to=nowhere,
   signal pointer angle=110,
]
\tw@declare@style*{menus}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@menus@base,signal to=east]%
      {\strut\CurrentMenuElement};%
}[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]%
{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@menus@base,signal from=west,signal to=east]%
      {\strut\CurrentMenuElement};%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@menus@base,signal from=west,]%
      {\strut\CurrentMenuElement};%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@menus@base]{\strut\CurrentMenuElement};%
}{gray}

\tikzstyle{tw@roundedmenus@base}=[%
   tw@set@tikz@colors,
   rounded corners=0.3ex,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
```

```
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
   signal,
   signal to=nowhere,
   signal pointer angle=110,
]
\tw@declare@style*{roundedmenus}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@roundedmenus@base,signal to=east]%
      {\strut\CurrentMenuElement};%
}[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]%
{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@roundedmenus@base,signal from=west,signal to=east]%
      {\strut\CurrentMenuElement};%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@roundedmenus@base,signal from=west,]%
      {\strut\CurrentMenuElement};%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@roundedmenus@base]{\strut\CurrentMenuElement};%
}{gray}

\tikzstyle{tw@angularmenus@base}=[%
   tw@set@tikz@colors,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
   signal,
   signal to=nowhere,
   signal pointer angle=110,
]
\tw@declare@style*{angularmenus}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@angularmenus@base,signal to=east]%
      {\strut\CurrentMenuElement};%
}[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]%
{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@angularmenus@base,signal from=west,signal to=east]%
      {\strut\CurrentMenuElement};%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@angularmenus@base,signal from=west,]%
      {\strut\CurrentMenuElement};%
```

```
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@angularmenus@base]{\strut\CurrentMenuElement};%
}{gray}

\tikzstyle{tw@roundedkeys@base}=[%
   tw@set@tikz@colors,
   rounded corners=0.3ex,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
]
\tw@declare@style@simple*{roundedkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@roundedkeys@base]{\strut\CurrentMenuElement};%
}[%
   \hspace{0.1em plus 0.1em minus 0.05em}%
   \raisebox{0.25ex}{\sffamily\footnotesize+}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{gray}

\tikzstyle{tw@shadowedroundedkeys@base}=[%
   tw@set@tikz@colors,
   rounded corners=0.3ex,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
   general shadow={%
      shadow xshift=.2ex, shadow yshift=-.15ex,
      fill=black,
   },
]
\tw@declare@style@simple*{shadowedroundedkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@shadowedroundedkeys@base]{\strut\CurrentMenuElement};%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \raisebox{0.25ex}{\sffamily\footnotesize+}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
][\hspace{0.2ex}]{gray}

\tikzstyle{tw@angularkeys@base}=[%
   tw@set@tikz@colors,
   inner sep=0pt,
```

```
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
]
\tw@declare@style@simple*{angularkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@angularkeys@base]{\strut\CurrentMenuElement};%
}[%
   \hspace{0.1em plus 0.1em minus 0.05em}%
   \raisebox{0.25ex}{\sffamily\footnotesize+}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{gray}

\tikzstyle{tw@shadowedangularkeys@base}=[%
   tw@set@tikz@colors,
   inner sep=0pt,
   inner xsep=2pt,
   text height=1.825ex,
   text depth=0.7ex,
   minimum width=1.5em,
   font=\relsize{-1}\sffamily,
   general shadow={%
      shadow xshift=.2ex, shadow yshift=-.15ex,
      fill=black,
   },
]
\tw@declare@style@simple*{shadowedangularkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]%
      \node(tw@node)[tw@shadowedangularkeys@base]{\strut\CurrentMenuElement};%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \raisebox{0.25ex}{\sffamily\footnotesize+}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
][\hspace{0.2ex}]{gray}

\tikzstyle{tw@typewriterkeys@base}=[%
   tw@set@tikz@colors,
   shape=circle,
   minimum size=2ex,
   inner sep=0.5pt, outer sep=1pt,
   font=\ttfamily\relsize{-1},
]
\tw@declare@style@simple*{typewriterkeys}{%
   \def\tw@typewriterkeys@curr@elem{%
      \maxsizebox*{2ex}{2ex}{\CurrentMenuElement}%
   }%
   \begin{tikzpicture}[baseline={($(tw@node.south)+(0,0.8ex)$)}]%
      \node(tw@node)[%
```

```
         tw@typewriterkeys@base, inner sep=1.25pt, line width=0.6pt%
      ]{\tw@typewriterkeys@curr@elem};
      \node[tw@typewriterkeys@base]{\tw@typewriterkeys@curr@elem};
   \end{tikzpicture}%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \raisebox{0.25ex}{\sffamily\footnotesize+}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{blacknwhite}

\tw@declare@style@simple*{paths}{%
   {\ttfamily\CurrentMenuElement}%
}[%
   \hspace{0.2em plus 0.1em}%
   \raisebox{0.08ex}{%
      \tikz\fill[\usemenucolor{txt}] (0,0) -- (0.5ex,0.5ex)%
                -- (0,1ex) -- cycle;%
   }%
   \hspace{0.2em plus 0.1em}%
]{blacknwhite}

\newcounter{tw@hyphen@char@num}
\newif\if@tw@hyphenatepaths@warnig
\@tw@hyphenatepaths@warnigtrue
\tw@declare@style@simple*{hyphenatepaths}{%
   {\ttfamily
    \IfStrEq{T1}{\encodingdefault}{%
       \setcounter{tw@hyphen@char@num}{23}%
    }{%
       \IfStrEq{OT1}{\encodingdefault}{%
          \setcounter{tw@hyphen@char@num}{255}%
       }{%
          \if@tw@hyphenatepaths@warnig%
          \tw@mk@warning{The hyphenatepaths styles will probably only\MessageBreak
          work with T1 or OT1 encoding.}%
          \fi\global\@tw@hyphenatepaths@warnigfalse%
       }%
    }%
    \hyphenchar\font=\value{tw@hyphen@char@num}\relax
    \CurrentMenuElement}%
}[%
   \hspace{0.2em plus 0.1em}%
   \raisebox{0.08ex}{%
      \tikz\fill[\usemenucolor{txt}] (0,0) -- (0.5ex,0.5ex)%
                -- (0,1ex) -- cycle;%
   }%
   \hspace{0.2em plus 0.1em}%
]{blacknwhite}

\newcommand*{\drawtikzfolder}[1][white]{%
```

```
   \begin{tikzpicture}[rounded corners=0.02ex,scale=0.7]
      \draw (0,0) -- (1em,0) -- (1em,1.5ex) -- (0.5em,1.5ex) -- %
            (0.4em,1.7ex) -- (0.1em,1.7ex) -- (0,1.5ex) -- cycle;
      \draw [fill=#1] (0,0) -- (1em,0) -- (0.85em,1.15ex) -- %
            ++(-1em,0) -- cycle;
   \end{tikzpicture}%
}

\copymenustyle{pathswithfolder}{paths}
\changemenuelement{pathswithfolder}{pre}{%
   \textcolor{\usemenucolor{txt}}{\drawtikzfolder}%
   \hspace{0.2em plus 0.1em}%
}

\copymenustyle{pathswithblackfolder}{paths}
\changemenuelement{pathswithblackfolder}{pre}{%
   \textcolor{\usemenucolor{txt}}{\drawtikzfolder[\usemenucolor{txt}]}%
   \hspace{0.2em plus 0.1em}%
}

\copymenustyle{hyphenatepathswithfolder}{hyphenatepaths}
\changemenuelement{hyphenatepathswithfolder}{pre}{%
   \textcolor{\usemenucolor{txt}}{\drawtikzfolder}%
   \hspace{0.2em plus 0.1em}%
}

\copymenustyle{hyphenatepathswithblackfolder}{hyphenatepaths}
\changemenuelement{hyphenatepathswithblackfolder}{pre}{%
   \textcolor{\usemenucolor{txt}}{\drawtikzfolder[\usemenucolor{txt}]}%
   \hspace{0.2em plus 0.1em}%
}
```

6.6 Menu macros

6.6.1 Internal commands

`\tw@default@input@sep` First we define our default input seperator

```
\edef\tw@default@input@sep{,}
```

`\CurrentMenuElement` and the `\CurrentMenuElement` dummy

```
\def\CurrentMenuElement{}
```

`\tw@define@menu@macro` Then we set up the internal command to create new menu macros. The list parsing code was essentially provided by Ahmed Musa at http://tex.stackexchange.com/a/44989/4918. Thank you very much!

```
\begingroup
\lccode`\,=1
\lowercase{\endgroup
  \robust@def*\tw@mk@test@input@sep#1{%
```

```
    \xifinsetTF{,\cpttrimspaces{#1},}{,bslash,backslash,directory,location,}%
  }%
}
\NewDocumentCommand{\tw@define@menu@macro}{%
   m O{\tw@default@input@sep} m
}{%
   \ifcsundef{tw@style@#3@sep}{%
      \tw@mk@error{Can't define menu macro \string#1\space,\MessageBreak
      because the style '#3' is not available!}
   }{%
      \csdef{tw@parse@menu@list@\expandafter\@gobble\string#1}##1{%
         \iflastindris
            \ifnum\indrisnr=\@ne
                \edef\CurrentMenuElement{##1}%
                \@nameuse{tw@style@#3@single}%
            \else
                \edef\CurrentMenuElement{##1}%
                \@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@last}%
            \fi
         \else
            \ifnum\indrisnr=\@ne
               \edef\CurrentMenuElement{##1}%
               \@nameuse{tw@style@#3@first}%
            \else
               \edef\CurrentMenuElement{##1}%
               \@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@mid}%
            \fi
         \fi
      }%
      \expandafter\newcommand\csname\expandafter\@gobble\string#1\endcsname[2][#2]{%
         {\def\tw@current@color@theme{\csname tw@style@#3@color@theme\endcsname}%
         \@nameuse{tw@style@#3@pre}%
         \tw@mk@test@input@sep{##1}{%
            \edef\tw@menu@list{\detokenize{##2}}\edef\tw@mk@tempa{\@backslashchar}%
         }{%
            \edef\tw@menu@list{\unexpanded{##2}}\edef\tw@mk@tempa{\cpttrimspaces{##1}}%
         }%
         {\letcs{\tw@mk@tempb}{tw@parse@menu@list@\expandafter\@gobble\string#1}%
         \cptexpanded{\indrisloop*[\tw@mk@tempa]}\tw@menu@list\tw@mk@tempb}%
         \@nameuse{tw@style@#3@post}}%
      }%
%      \expandafter\cptrobustify\csname\expandafter\@gobble\string#1\endcsname%
   }%
}
\edef\cpt@parserlist{\cpt@parserlist\@backslashchar}
```

6.6.2 User-level commands

\newmenumacro
\renewmenumacro
\providemenumacro

Now it's time to build the user-level commands

```
\NewDocumentCommand{\newmenumacro}{m O{\tw@default@input@sep} m}{%
   \ifcsundef{\expandafter\@gobble\string#1}{%
      \tw@define@menu@macro{#1}[#2]{#3}%
      \expandafter\cptrobustify\csname\expandafter\@gobble\string#1\endcsname%
   }{
      \tw@mk@error{Menu macro '\string#1' already defined!\MessageBreak
      Use \string\renewmenustyle\space instead.}
   }%
}
\NewDocumentCommand{\renewmenumacro}{m O{\tw@default@input@sep} m}{%
   \cslet{\expandafter\@gobble\string#1}{\relax}%
   \tw@define@menu@macro{#1}[#2]{#3}%
}
\NewDocumentCommand{\providemenumacro}{m O{\tw@default@input@sep} m}{%
   \ifcsundef{\expandafter\@gobble\string#1}{%
      \tw@define@menu@macro{#1}[#2]{#3}%
   }{
      \tw@mk@warning{Menu macro '\string#1' already defined!\MessageBreak
      Use \string\renewmenustyle\space to redefine it.}
   }%
}
```

6.6.3 Predefined menu macros

Now we got all tools to predefine some menu macros. To be sure that these commands won't conflict with other packages we introduced the option `definemacros`. Here we have to check it:

```
\iftw@mk@definemenumacros
```

`\menu` `\directory` `\keys`

And then we define three basic macros.

```
\newmenumacro{\menu}[>]{menus}
\newmenumacro{\directory}[/]{paths}
\newmenumacro{\keys}[+]{roundedkeys}
```

Lastly we close the `definemacros` if statment:

```
\fi
```

6.7 Keys

Before we define anything we check if the user allows it:

```
\iftw@mk@definekeys
```

Before define the key macros we create some macros that save some typing by condensing the similarities between the key marcros.

`\tw@make@key@box`

The first of these macros helps us building save boxes to store the `{tikzpicture}`, that will draw the key later. This is necessary because otherwise the picture will inherit the style of the key sequence `node`.

```
\NewDocumentCommand{\tw@make@key@box}{m m}{%
```

```
    \expandafter\newbox\csname tw@mk@box@#1\endcsname
    \expandafter\sbox\csname tw@mk@box@#1\endcsname{%
       #2%
    }%
    \csdef{tw@mk@#1}{%
       \expandafter\usebox\csname tw@mk@box@#1\endcsname%
    }%
}
```

`\tw@make@key@macro` The next macro defines the user level command by accessing a macro like `tw@mk@`⟨*key*⟩ or `tw@mk@`⟨*key*⟩`@`⟨*os*⟩, if the apperance differs between Mac and Windows. To use this macro we assume that the `tw@mk@`⟨*key*⟩ commads are defined.

```
\NewDocumentCommand{\tw@make@key@macro}{s m}{%
    \IfBooleanTF{#1}{%
       \expandafter\providecommand\csname\expandafter\@gobble\string#2\endcsname{%
          \expandonce{\maxsizebox{!}{1.8ex}{%
             \@nameuse{tw@mk@\expandafter\@gobble\string#2@\tw@mk@os}}%
          }%
       }%
       \expandafter\providecommand\csname\expandafter\@gobble\string#2mac\endcsname{%
          \expandonce{\maxsizebox{!}{1.8ex}{%
             \@nameuse{tw@mk@\expandafter\@gobble\string#2@mac}}%
          }%
       }%
       \expandafter\providecommand\csname\expandafter\@gobble\string#2win\endcsname{%
          \expandonce{\maxsizebox{!}{1.8ex}{%
             \@nameuse{tw@mk@\expandafter\@gobble\string#2@win}}%
          }%
       }%
    }{%
       \expandafter\providecommand\csname\expandafter\@gobble\string#2\endcsname{%
          \expandonce{\maxsizebox{!}{1.8ex}{%
             \@nameuse{tw@mk@\expandafter\@gobble\string#2}}%
          }%
       }%
    }%
}
```

`\tw@define@mackey` The last helping macro is `\tw@define@mackey`. We use it to exequte code depending on the `mackeys` option.

```
\newcommand*{\tw@define@mackey}[2]{%
    \IfStrEq{text}{\tw@mk@mackeys}{#1}{%
       \IfStrEq{symbols}{\tw@mk@mackeys}{#2}{}%
    }%
}
```

`\shift` `\capslock` `\tab` `\esc` `\ctrl` `\alt` `\AltGr` `\cmd` `\Space` `\return` `\enter` `\winmenu` `\backspace` Now we ar prepared to generate the key macros. I will be nearly the same way for all keys. Step one is to build a `tw@mk@`⟨*key*⟩ macro

```
\tw@make@key@box{shift}{%
```

```
   \begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
      \draw (0.3ex,0) -- (1.1ex,0) -- (1.1ex,1.2ex) -- %
            (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
            (0.3ex,1.2ex) -- cycle;
   \end{tikzpicture}%
}
```

and then we define the user-level command \⟨*key*⟩

```
\tw@make@key@macro{\shift}
```

It's a little more complicated if the apperance should differ depending on the OS. The first step again is to define `tw@mk@`⟨*key*⟩`@mac` and `tw@mk@`⟨*key*⟩`@win`:

```
\tw@make@key@box{capslock@mac}{%
   \begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
      \draw (0.3ex,0.7ex) -- (1.1ex,0.7ex) -- (1.1ex,1.2ex) -- %
            (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
            (0.3ex,1.2ex) -- cycle;
      \draw (0.3ex,0) rectangle (1.1ex,0.4ex);
   \end{tikzpicture}%
}
\tw@make@key@box{capslock@win}{%
   \begin{tikzpicture}[yscale=-1,yshift=-1.8ex,baseline={(0,0)},semithick]
      \draw (0.3ex,0) -- (1.1ex,0) -- (1.1ex,1.2ex) -- %
            (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
            (0.3ex,1.2ex) -- cycle;
   \end{tikzpicture}%
}
```

And then use the starred version `\tw@make@key@macro*` which creates \⟨*key*⟩ that depends on the `os` option, \⟨*key*⟩`mac` and \⟨*key*⟩`win`, that are not affected by `os`.

```
\tw@make@key@macro*{\capslock}
```

Here are the other macros:

```
\tw@make@key@box{tab@mac}{%
   \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)}]
      \draw [->,semithick] (0,0) -- (1em,0);
      \draw (1em,-0.45ex) -- (1em,0.45ex);
   \end{tikzpicture}%
}
\tw@make@key@box{tab@win}{%
   \begin{tikzpicture}[yshift=0.1ex,baseline={(0,0)}]
      \draw [->,semithick] (0.2em,0) -- (1.2em,0);
      \draw (1.2em,-0.45ex) -- (1.2em,0.45ex);
      \draw [<-,semithick] (0,1ex) -- (1em,1ex);
      \draw (0,0.55ex) -- (0,1.55ex);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\tab}

\def\tw@mk@esc@win{Esc}
\tw@define@mackey{%
```

```
   \def\tw@mk@esc@mac{esc}
}{%
   \tw@make@key@box{esc@mac}{%
      \begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
         \draw [->] (0.5ex,0.5ex) -- ++(45:1.5ex);
         \draw (0.5ex,0.5ex) ++(15:0.6ex) arc (15:-285:0.6ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\esc}

\providecommand\ctrlname{Ctrl}
\def\tw@mk@ctrl@win{\ctrlname}
\def\tw@mk@ctrl@mac{ctrl}
\tw@make@key@macro*{\ctrl}

\def\tw@mk@Alt@win{Alt}
\tw@define@mackey{%
   \def\tw@mk@Alt@mac{alt}%
}{%
   \tw@make@key@box{Alt@mac}{%
      \begin{tikzpicture}[semithick]
         \draw (0,1ex) -- (0.5ex,1ex) -- (1ex,0.3ex) -- (1.8ex,0.3ex);
         \draw (0.8ex,1ex) -- (1.8ex,1ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\Alt}

\providecommand*{\AltGr}{Alt\,Gr}

\def\tw@mk@cmd@win{%
   \tw@mk@warning{'\string\cmd' only for Mac!}%
}
\tw@define@mackey{%
   \def\tw@mk@cmd@mac{cmd}%
}{%
   \tw@make@key@box{cmd@mac}{%
      \begin{tikzpicture}[yshift=-0.15ex,baseline={(0,0)},semithick]
         \draw (0.5ex,0.7ex) -- (0.5ex,1.25ex) arc (0:270:0.25ex) -- %
               (1.25ex,1ex) arc (-90:180:0.25ex) -- (1ex,0.25ex) %
               arc (-180:90:0.25ex) -- (0.25ex,0.5ex) arc (90:360:0.25ex) %
               -- cycle;
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\cmd}

\providecommand*{\Space}{\expandonce{\rule{3em}{0pt}}}

```

```
\tw@make@key@box{return@mac}{%
   \begin{tikzpicture}[semithick]
      \draw [->, rounded corners=0.3ex] (1.25ex,1ex) -| %
            (2ex,0) -- (0,0);
   \end{tikzpicture}%
}
\tw@make@key@box{return@win}{%
   \begin{tikzpicture}[semithick]
      \draw [->] (1ex,1.25ex) |- (0,0);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\return}

\def\tw@mk@enter@win{Enter}
\tw@make@key@box{enter@mac}{%
   \begin{tikzpicture}[semithick]
      \draw (0,0) -- (0.5ex,0.5ex) -- (1ex,0);
      \draw (0,0.55ex) -- (1ex,0.55ex);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\enter}

\def\tw@mk@winmenu@mac{%
   \tw@mk@warning{'\string\winmenu' only for Windows!}%
}
\tw@make@key@box{winmenu@win}{%
   \begin{tikzpicture}[yshift=-0.2ex,baseline={(0,0)},semithick]
      \draw (0,0) rectangle (1.5ex,1.8ex);
      \draw (0.25ex,1.4ex) -- ++(1ex,0);
      \draw (0.25ex,1ex) -- ++(1ex,0);
      \draw (0.25ex,0.6ex) -- ++(1ex,0);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\winmenu}

\tw@make@key@box{backspace}{%
   \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)},thick]
      \draw [<-] (0,0) -- (1.25em,0);
   \end{tikzpicture}%
}
\tw@make@key@macro{\backspace}

\providecommand{\delname}{Del.}
\def\tw@mk@del@win{\delname}
\tw@define@mackey{%
   \def\tw@mk@del@mac{\delname}%
}{%
   \tw@make@key@box{del@mac}{%
      \begin{tikzpicture}
         \draw [semithick] (0,0) -- (1.5ex,0) -- (2ex,0.5ex) --%
```

```
               (1.5ex,1ex) -- (0,1ex) -- cycle;
         \draw (0.5ex,0.2ex) -- (1.1ex,0.8ex);
         \draw (0.5ex,0.8ex) -- (1.1ex,0.2ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\del}
```

\arrowkeyup
\arrowkeydown
\arrowkeyleft
\arrowkeyright

Lastly we define the arrow macros:

```
\tw@make@key@box{arrowkeyup}{%
   \begin{tikzpicture}[yshift=-0.2ex,baseline={(0,0)}]
      \draw [->,semithick] (0,0) -- (0,0.8em);
   \end{tikzpicture}%
}
\tw@make@key@macro{\arrowkeyup}

\tw@make@key@box{arrowkeydown}{%
   \begin{tikzpicture}[yshift=0.7em,baseline={(0,0)}]
      \draw [->,semithick] (0,0) -- (0,-0.8em);
   \end{tikzpicture}%
}
\tw@make@key@macro{\arrowkeydown}

\tw@make@key@box{arrowkeyright}{%
   \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)}]
      \draw [->,semithick] (0,0) -- (0.8em,0);
   \end{tikzpicture}%
}
\tw@make@key@macro{\arrowkeyright}

\tw@make@key@box{arrowkeyleft}{%
   \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)}]
      \draw [->,semithick] (0,0) -- (-0.8em,0);
   \end{tikzpicture}%
}
\tw@make@key@macro{\arrowkeyleft}
```

\arrowkey

And the \arrowkey macro that get's it's direction as argument.

```
\newcommand{\arrowkey}[1]{%
   \IfStrEq{^}{#1}{\arrowkeyup}{%
      \IfStrEq{v}{#1}{\arrowkeydown}{%
         \IfStrEq{<}{#1}{\arrowkeyleft}{%
            \IfStrEq{>}{#1}{\arrowkeyright}{%
               \tw@mk@error{Wrong value '#1' for \string\arrowkey\MessageBreak
               Possible values are '^', 'v', '<' or '>'}%
            }%
         }%
      }%
   }%
}
```

Close the `\iftw@mk@definekeys`

```
\fi
```