MENUKEYS

Tobias Weh
mail@tobiw.de
https://tobiw.de/en

Jonathan P. Spratte
jspratte@yahoo.de

https://github.com/tweh/menukeys
https://www.ctan.org/pkg/menukeys
macros▸latex▸contrib▸menukeys

2026/05/14 — v1.6.3

Abstract

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

You're welcome to send me feedback, questions, bug reports and feature requests. 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 original list parsing code at https://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. Most predefined styles use the power of TikZ[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 millimeters, inches and pixels. The shortcut
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 millimeters, inches and pixels. The shortcut 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 https://www.ctan.org/pkg/pgf.

3 Package loading and options

Since `menukeys` used to use `catoptions`, which does some heavy changes on key-value options, it **was** recommended to load `menukeys` as the last package (even after `hyperref`[2]). **This is no longer the case!**

These are the possible options:

`definemenumacros` (opt.)

definemenumacros: Most of `menukeys`' macros should not conflict with other packages[3] 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 they conflict with other packages. To prevent this you can 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 4.4 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 whether 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 differently. The default value is `mac`.

hyperrefcolorlinks: *Obsolete* (see sec. 5 and 6.4.1).

4 Usage

4.1 Basics

`menukeys` comes with three "menu macros" that parse and print lists. We have `\menu` `\menu{`⟨*menu sequence*⟩`}`, with `>` as default input list separator, `\directory` `\directory{`⟨*path and files*⟩`}` with `/` as default separator and `\keys` `\keys{`⟨*keystrokes*⟩`}` with `+` as default separator. You've seen examples for all of them in section 1.

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*⟩`}`.[4]

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 separator will be ignored, i.e. `\keys{\ctrl+C}` equals `\keys{\ctrl + C}`.

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

Extras › Settings › Units, rulers and origin

[2] See https://tex.stackexchange.com/q/237683/4918 and https://github.com/tweh/menukeys/issues/41.

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

[4] If you want to change the input separator globally it's recommended to renew the menu macro as described in section 4.4.

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.

Both pre-defined and custom styles may be applied using the command `\renewmenumacro{`⟨*macro*⟩`}[`⟨*input sep*⟩`]{`⟨*style*⟩`}`. For further information on defining and using macro styles, refer to section 4.4. For example, the "shadowed angular" keys style may be applied using the command `\renewmenumacro{\keys}{shadowedangularkeys}`.

4.2.1 Predefined styles

Name: `menus`

File › Extras › Preferences

Menu

```
\newmenumacro\test{menus}
\test{File,Extras,Preferences}
\test{Menu}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This File › Extras › Preferences is the result of a style which name is `menus`. And again some blind text without any sense.

This is the default style for `\menu`.

Name: `roundedmenus`

File › Extras › Preferences

Menu

```
\renewmenumacro\test{roundedmenus}
\test{File,Extras,Preferences}
\test{Menu}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This File › Extras › Preferences is the result of a style which name is `roundedmenus`. And again some blind text without any sense.

Name: `angularmenus`

File › Extras › Preferences

Menu

```
\renewmenumacro\test{angularmenus}
\test{File,Extras,Preferences}
\test{Menu}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This File › Extras › Preferences is the result of a style which name is `angularmenus`. And again some blind text without any sense.

Name: `roundedkeys`

```
\renewmenumacro\test{roundedkeys}
\test{Ctrl,Alt,Q}
\test{S}
```

Ctrl + Alt + Q

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This Ctrl + Alt + Q is the result of a style which name is `roundedkeys`. And again some blind text without any sense.

The color of + is taken from optional color B.
This is the default style for `\keys`*.*

Name: `shadowedroundedkeys`

```
\renewmenumacro\test{shadowedroundedkeys}
\test{Ctrl,Alt,Q}
\test{S}
```

Ctrl + Alt + Q

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This Ctrl + Alt + Q is the result of a style which name is `shadowedroundedkeys`. And again some blind text without any sense.

The color of + is taken from optional color B.
The shadow color is taken from optional color C.

Name: `angularkeys`

```
\renewmenumacro\test{angularkeys}
\test{Ctrl,Alt,Q}
\test{S}
```

Ctrl + Alt + Q

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This Ctrl + Alt + Q is the result of a style which name is `angularkeys`. And again some blind text without any sense.

The color of + is taken from optional color B.

Name: `shadowedangularkeys`

```
\renewmenumacro\test{shadowedangularkeys}
\test{Ctrl,Alt,Q}
\test{S}
```

Ctrl + Alt + Q

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This Ctrl + Alt + Q is the result of a style which name is `shadowedangularkeys`. And again some blind text without any sense.

The color of + is taken from optional color B.
The shadow color is taken from optional color C.

Name: typewriterkeys

```
\renewmenumacro\test{typewriterkeys}
\test{Alt,Q}
\test{S}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This is the result of a style which name is typewriterkeys. And again some blind text without any sense.

The color of + is taken from optional color B.

Name: paths

C:‣User‣Folder‣MyFile.tex

MyFile.tex

```
\renewmenumacro\test{paths}
\test{C:,User,Folder,MyFile.tex}
\test{MyFile.tex}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This C:‣User‣Folder‣MyFile.tex is the result of a style which name is paths. And again some blind text without any sense.

The sep color is taken from optional color C.
This is the default style for `\directory`*.*

Name: pathswithfolder

C:‣User‣Folder‣MyFile.tex

MyFile.tex

```
\renewmenumacro\test{pathswithfolder}
\test{C:,User,Folder,MyFile.tex}
\test{MyFile.tex}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This C:‣User‣Folder‣MyFile.tex is the result of a style which name is pathswithfolder. And again some blind text without any sense.

The folder draw color is taken from optional color B.
The folder fill color is taken from optional color A.
The sep color is taken from optional color C.

Name: pathswithblackfolder

C:‣User‣Folder‣MyFile.tex

MyFile.tex

```
\renewmenumacro\test{pathswithblackfolder}
\test{C:,User,Folder,MyFile.tex}
\test{MyFile.tex}
```

This is some more or less blind text, to demonstrate how the sequence looks in text. This C:‣User‣Folder‣MyFile.tex is the result of a style which name is pathswithblackfolder. And again some blind text without any sense.

The folder draw color is taken from optional color B.
The folder fill color is taken from optional color C.
The sep color is taken from optional color C.

The following three styles allow paths elements to be hyphenated, 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) and that this in some cases doesn't work very well.

Name: hyphenatepaths

C: ‣ Database ‣ User ‣ ALongUser NameHere ‣ ALongerFolder NameAtThisPlace‣MyFile.tex

MyFile.tex

This is some more or less blind text, to demonstrate how the sequence looks in text. This C: ‣ Database ‣ User ‣ ALongUserNameHere ‣ ALongerFolder NameAtThisPlace ‣ MyFile.tex is the result of a style which name is hyphenatepaths. And again some blind text without any sense.

The sep color is taken from optional color C.

```
\renewmenumacro\test{hyphenatepaths}
\test{C:,Database,User,ALongUserNameHere,
  ALongerFolderNameAtThisPlace,MyFile.tex}
\test{MyFile.tex}
```

Name: hyphenatepathswithfolder

🗀 C: ‣ Database ‣ User ‣ ALongUser NameHere ‣ ALongerFolder NameAtThisPlace‣MyFile.tex

🗀 MyFile.tex

This is some more or less blind text, to demonstrate how the sequence looks in text. This 🗀 C: ‣ Database ‣ User ‣ ALongUserNameHere ‣ ALongerFolder NameAtThisPlace ‣ MyFile.tex is the result of a style which name is hyphenatepathswithfolder. And again some blind text without any sense.

The folder draw color is taken from optional color B.
The folder fill color is taken from optional color A.
The sep color is taken from optional color C.

```
\renewmenumacro\test
  {hyphenatepathswithfolder}
\test{C:,Database,User,ALongUserNameHere,
  ALongerFolderNameAtThisPlace,MyFile.tex}
\test{MyFile.tex}
```

Name: hyphenatepathswithblackfolder

🗀 C: ‣ Database ‣ User ‣ ALongUser NameHere ‣ ALongerFolder NameAtThisPlace‣MyFile.tex

🗀 MyFile.tex

This is some more or less blind text, to demonstrate how the sequence looks in text. This 🗀 C: ‣ Database ‣ User ‣ ALongUserNameHere ‣ ALongerFolder NameAtThisPlace ‣ MyFile.tex is the result of a style which name is hyphenatepathswithblackfolder. And again some blind text without any sense.

The folder draw color is taken from optional color B.
The folder fill color is taken from optional color C.
The sep color is taken from optional color C.

```
\renewmenumacro\test
  {hyphenatepathswithblackfolder}
\test{C:,Database,User,ALongUserNameHere,
  ALongerFolderNameAtThisPlace,MyFile.tex}
\test{MyFile.tex}
```

\drawtikzfolder **Hint** The folder is drawn with the command `\drawtikzfolder` which is part of `menukeys` and has two optional arguments to change the color of the lines and the fill color of the front:
`\drawtikzfolder[`⟨*front fill*⟩`][`⟨*draw*⟩`]`

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 4.3).

Example Let us consider we want a list that prints a frame around its elements and separates 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 style.

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

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

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

theme is a color theme (see section 4.3).

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 your 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 4.3 for details about possible elements).

4.2.3 Copying styles

`\copymenustyle` To copy an existing style to a new style use `\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` from 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 redefining 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 become 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, and additionally two optional colors used by some themes.

4.3.1 Predefined themes

There are two predefined color themes

Name: `gray`

Background: ☐ Border: ■ Text: ■ (A: ☐ B: ■ C: ■)

Name: `blacknwhite`

Background: ☐ Border: ■ Text: ■ (A: ☐ B: ■ C: ■)

4.3.2 Create a theme

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

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.

a is an optional additional color (by default same as bg).

b is an optional additional color (by default same as br).

c is an optional additional color (by default same as txt).

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 separated by a special symbol to print it with a menu style.

4.4.1 Predefined menu macros

See section 4.1.

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 separator used in the input list (see section 4.1 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 work like the LaTeX macros `\renewcommand` and `\providecommand`.

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

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

4.5 Keys

The `menukeys` package comes with some macros to print special keys in the sequences set with `\keys`. Depending on the given OS (see section 3) 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.

The full ist of key macros is shown in table 1.

Table 1: Overview of all key macros.

Macro	Mac	Win.
`\shift`	⇧	⇧
`\capslock`	⇪	⇩
`\tab`	⭾	↹
`\esc`	esc / ⎋	Esc
`\oldesc`	esc / ⎋	Esc
`\ctrl`	ctrl	Ctrl
`\Alt`	alt / ⌥	Alt
`\AltGr`		Alt Gr
`\cmd`	cmd / ⌘	
`\Space`	[empty sp.]	[empty sp.]
`\SPACE`	Space	Space
`\return`	⏎	↵
`\enter`	⌅	Enter

Macro	Mac	Win.
`\winmenu`		▤
`\backspace`	⟵	⟵
`\del`	Del. / ⌦	Del.
`\backdel`	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` `\spacename` The texts for `\ctrl`, `\del` and `\SPACE` are saved in `\ctrlname`, `\delname`, `\spacename` 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 (left: `text`, right: `symbols`).

I apologize 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` or directly to it via `\PassOptionsToPackage`.

  Example Set `xcolor` to `cmyk` model:

  ```
  \documentclass{article}
  ```

```
\PassOptionsToPackage{cmyk}{xcolor}
\usepackage{menukeys}
\begin{document}
   Hello World!
\end{document}
```

If you find something to add to this list please send me an e-mail or report a bug on GitHub (https://github.com/tweh/menukeys).

6 Implementation

1 ⟨*pkg⟩

6.1 Required packages

Load the required packages

```
2 \RequirePackage{xparse}
3 \RequirePackage{xstring}
4 \RequirePackage{etoolbox}
```

Furthermore we need Ti*k*Z and some of its libraries,

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

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

```
7 \RequirePackage{xcolor}
8 \RequirePackage{adjustbox}
```

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

```
9 \RequirePackage{relsize}
```

To define the list parsing commands and allow `\` as a separator we used to load `catoptions`. Instead we now use some `expl3` functions to replace the macros we required from `catoptions`.

The first few of these functions are more or less direct equivalents. A bit of attention has to be paid for `\tw@mk@xifinsetTF` as it requires the arguments to get swapped.

```
10 \ExplSyntaxOn
11 \cs_new_eq:NN \tw@mk@trimspaces \tl_trim_spaces:n
12 \cs_new_eq:NN \tw@mk@exp@Nnno   \exp_args:Nnno
13 \cs_new_eq:NN \tw@mk@string     \cs_to_str:N
14 \prg_generate_conditional_variant:Nnn \tl_if_in:nn { xx } { TF }
15 \cs_new:Npn \tw@mk@xifinsetTF #1 #2
16   {
17     \tl_if_in:xxTF {#2} {#1}
18   }
```

The replacement for `\indrisloop` will not set the conditional `\iflastindris`, instead we can check whether the sequence is empty to see whether this is the last element. This test will not use a TeX-like `\iflastindris...\else...\fi` construct but instead two branches.

```
19 \cs_new:Npn \tw@mk@iflastindris
20   {
21     \seq_if_empty:NTF \l__twmk_indris_seq
22   }
```

Replacing `\indrisloop` is a bit more work as it requires us to push some values to a stack (to allow nested usage, this may not be necessary for `menukeys`, but it is part of the original `\indrisloop` so we should play nice here). First we'll need a few variables.

```
\seq_new:N \l__twmk_indris_seq
\int_new:N \l__twmk_indris_int
\tl_new:N  \l__twmk_indris_tl
\cs_new_eq:NN \tw@mk@indrisnr \l__twmk_indris_int
\seq_new:N \l__twmk_indris_seqstack_seq
\seq_new:N \l__twmk_indris_intstack_seq
```

Our stack will use another sequence in which the definitions of the parent call will be stored for the sequence and the integer. The other variables put on a stack by `\indrisloop` aren't required. The synopsis of `\tw@mk@indrisloop` will be different to the one provided by `catoptions`. The delimiter will be a mandatory argument (not in brackets), and there is no starred version.

```
\cs_new_protected:Npn \__twmk_pushseq:
  {
    \seq_push:No \l__twmk_indris_seqstack_seq \l__twmk_indris_seq
  }
\cs_new_protected:Npn \__twmk_pushint:
  {
    \seq_push:NV \l__twmk_indris_intstack_seq \l__twmk_indris_int
  }
\cs_new_protected:Npn \__twmk_popseq:
  {
    \seq_if_empty:NTF \l__twmk_indris_seqstack_seq
      { \seq_clear:N \l__twmk_indris_seq }
      { \seq_pop:NN \l__twmk_indris_seqstack_seq \l__twmk_indris_seq }
  }
\cs_new_protected:Npn \__twmk_popint:
  {
    \seq_if_empty:NTF \l__twmk_indris_intstack_seq
      { \int_zero:N \l__twmk_indris_int }
      {
        \group_begin:
          \seq_pop:NN \l__twmk_indris_intstack_seq \l__twmk_indris_tl
          \exp_args:NNNo
        \group_end:
        \int_set:Nn \l__twmk_indris_int \l__twmk_indris_tl
      }
  }
```

The real loop works by first splitting the input into a sequence according to the delimiter in `#1`. Then this sequence is stepped through, but instead of using `\seq_map:NN` we'll have to pop the sequence into a local variable so that our test for the last element works. The parameter `#2` has to be expanded once as it is handed in as a token storing the real argument in later use.

```
\cs_generate_variant:Nn \seq_set_split:Nnn { Noo }
\cs_new_protected:Npn \tw@mk@indrisloop #1 #2 #3
  {
    \__twmk_pushseq:
    \__twmk_pushint:
    \seq_set_split:Noo \l__twmk_indris_seq {#1} {#2}
```

```
    \int_zero:N \l__twmk_indris_int
    \bool_do_while:nn { \bool_not_p:n { \seq_if_empty_p:N \l__twmk_indris_seq } }
      {
        \int_incr:N \l__twmk_indris_int
        \seq_pop_left:NN \l__twmk_indris_seq \l__twmk_indris_tl
        \exp_args:No #3 \l__twmk_indris_tl
      }
    \__twmk_popseq:
    \__twmk_popint:
  }
\ExplSyntaxOff
```

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 commands 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}
\DeclareBoolOption[false]{hyperrefcolorlinks}
\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 Workarounds

Some workarounds to "solve" some incompatibilities:

6.4.1 `hyperref`'s `colorlinks` option

There used to be an issue with using the `colorlinks` option of `hyperref` due to `catoptions` being loaded. Since `catoptions` isn't required any more, this workaround isn't necessary. For backwards compatibility the `hyperrefcolor links` option is still evaluated and just uses `\hypersetup` or `\PassOptionsToPackage` depending on whether `hyperref` is already loaded.

```
\iftw@mk@hyperrefcolorlinks
  \tw@mk@warning{The option 'hyperrefcolorlinks' is obsolete}
  \@ifpackageloaded{hyperref}
    {\hypersetup{colorlinks}}
    {\PassOptionsToPackage{colorlinks}{hyperref}}
\fi
```

6.5 Color themes

6.5.1 Internal commands

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

```
\newcommand*{\tw@make@color@theme}[8]{%
    \definecolor{tw@color@theme@#1@bg}{#2}{#3}%
    \definecolor{tw@color@theme@#1@br}{#2}{#4}%
    \definecolor{tw@color@theme@#1@txt}{#2}{#5}%
    \definecolor{tw@color@theme@#1@a}{#2}{#6}%
    \definecolor{tw@color@theme@#1@b}{#2}{#7}%
    \definecolor{tw@color@theme@#1@c}{#2}{#8}%
}
```

6.5.2 User-level commands

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

`\renewmenucolortheme`

```
\NewDocumentCommand{\newmenucolortheme}{ m m m m m O{#3} O{#4} O{#5} }{%
    \@ifundefinedcolor{tw@color@theme@#1@bg}{%
        \tw@make@color@theme{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}%
    }{%
        \tw@mk@error{Color theme '#1' already defined!\MessageBreak
        Use \string\renewmenucolortheme\space instead.}%
    }
}
\NewDocumentCommand{\renewmenucolortheme}{ m m m m m O{#3} O{#4} O{#5} }{%
    \tw@make@color@theme{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}%
}
```

\changemenucolor Lastly 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}%
      \colorlet{tw@color@theme@#1@a}{tw@color@theme@#2@a}%
      \colorlet{tw@color@theme@#1@b}{tw@color@theme@#2@b}%
      \colorlet{tw@color@theme@#1@c}{tw@color@theme@#2@c}%
   }{%
      \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%
}
```

6.5.3 Predefined themes

There are two predefined color themes

```
\newmenucolortheme{gray}{gray}{0.95}{0.3}{0}[0.95][0][0]
\newmenucolortheme{blacknwhite}{gray}{1}{0}{0}[1][0][0]
```

6.6 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 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}{}
```

6.6.1 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@style
\tw@declare@style@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@style@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@style@extra@args%
}
```

6.6.2 User-level commands

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}%
   }%
}
```

6.6.3 Copying and changing

\copymenustyle The last two steps in this part are to define a command 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\color{\usemenucolor{txt}}\CurrentMenuElement};}}%
            }%
         }%
      }{\tw@mk@error{No element '#3'. Possible values are\MessageBreak
        single, first, middle, last, pre, post or sep.}}%
   }%
}
```

6.6.4 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

```
\tikzset{tw@set@tikz@colors/.style={%
   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 `nodes`.

```
\tikzset{tw@menus@base/.style={%
   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\color{\usemenucolor{txt}}\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\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@menus@base,signal from=west,]%
      {\strut\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@menus@base]{\strut\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{gray}

\tikzset{tw@roundedmenus@base/.style={%
   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\color{\usemenucolor{txt}}\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\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@roundedmenus@base,signal from=west,]%
      {\strut\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@roundedmenus@base]{\strut\color{\usemenucolor{txt}}\CurrentMenuElemen
```

```
}{gray}

\tikzset{tw@angularmenus@base/.style={%
   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\color{\usemenucolor{txt}}\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\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@angularmenus@base,signal from=west,]%
      {\strut\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@angularmenus@base]{\strut\color{\usemenucolor{txt}}\CurrentMenuElemen
}{gray}

\tikzset{tw@roundedkeys@base/.style={%
   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\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}[%
   \hspace{0.1em plus 0.1em minus 0.05em}%
   \textcolor{\usemenucolor{b}}{\raisebox{0.25ex}{\sffamily\relsize{-2}+}}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{gray}

\tikzset{tw@shadowedroundedkeys@base/.style={%
   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=\usemenucolor{c},
   },
}}
\tw@declare@style@simple*{shadowedroundedkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@shadowedroundedkeys@base]%
         {\strut\color{\usemenucolor{txt}}\CurrentMenuElement};%
   }%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \textcolor{\usemenucolor{b}}{\raisebox{0.25ex}{\sffamily\relsize{-2}+}}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
⟨/pkg⟩
⟨cur⟩][\kern0.2ex\relax]{gray}
⟨161⟩][\hspace{0.2ex}]{gray}
⟨*pkg⟩

\tikzset{tw@angularkeys@base/.style={%
   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\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}[%
   \hspace{0.1em plus 0.1em minus 0.05em}%
   \textcolor{\usemenucolor{b}}{\raisebox{0.25ex}{\sffamily\relsize{-2}+}}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{gray}

\tikzset{tw@shadowedangularkeys@base/.style={%
   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=\usemenucolor{c},
   },
}}
\tw@declare@style@simple*{shadowedangularkeys}{%
   \tikz[baseline={($(tw@node.base)+(0,-0.2ex)$)}]{%
      \node(tw@node)[tw@shadowedangularkeys@base]%
         {\strut\color{\usemenucolor{txt}}\CurrentMenuElement};}%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \textcolor{\usemenucolor{b}}{\raisebox{0.25ex}{\sffamily\relsize{-2}+}}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
⟨/pkg⟩
⟨cur⟩][\kern0.2ex\relax]{gray}
⟨161⟩][\hspace{0.2ex}]{gray}
⟨*pkg⟩

\tikzset{tw@typewriterkeys@base/.style={%
   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%
      ]{\color{\usemenucolor{txt}}\tw@typewriterkeys@curr@elem};
      \node[tw@typewriterkeys@base]%
         {\color{\usemenucolor{txt}}\tw@typewriterkeys@curr@elem};
   \end{tikzpicture}%
}[%
   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
   \textcolor{\usemenucolor{b}}{\raisebox{0.25ex}{\sffamily\relsize{-2}+}}%
   \hspace{0.1em plus 0.1em minus 0.05em}%
]{blacknwhite}

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

\NewDocumentCommand{\drawtikzfolder}{O{white} O{black}}{%
   \begin{tikzpicture}[rounded corners=0.02ex,scale=0.7]
      \draw [#2] (0,0) -- (1em,0) -- (1em,1.5ex) -- (0.5em,1.5ex) -- %
            (0.4em,1.7ex) -- (0.1em,1.7ex) -- (0,1.5ex) -- cycle;
      \draw [#2,fill=#1] (0,0) -- (1em,0) -- (0.85em,1.15ex) -- %
            ++(-1em,0) -- cycle;
   \end{tikzpicture}%
}

\copymenustyle{pathswithfolder}{paths}
\changemenuelement{pathswithfolder}{pre}{%
   \drawtikzfolder[\usemenucolor{a}][\usemenucolor{b}]%
   \hspace{0.2em plus 0.1em}%
}

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

\copymenustyle{hyphenatepathswithfolder}{hyphenatepaths}
\changemenuelement{hyphenatepathswithfolder}{pre}{%
   \drawtikzfolder[\usemenucolor{a}][\usemenucolor{b}]%
   \hspace{0.2em plus 0.1em}%
```

```
}

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

6.7 Menu macros

6.7.1 Internal commands

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

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

`\CurrentMenuElement` and the `\CurrentMenuElement` dummy

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

`\tw@define@menu@macro` `\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 https://tex.stackexchange.com/a/44989/4918. Jonathan P. Spratte made some major changes to make `menukeys` work without `catoptions` and reimplemented the parsing code using LaTeX3. Thank you both very much!

```
\begingroup
\lccode`\,=1
\lowercase{\endgroup
  \@ifdefinable\tw@mk@test@input@sep
    {%
      \protected\def\tw@mk@test@input@sep#1{%
        \tw@mk@xifinsetTF
          {,\tw@mk@trimspaces{#1},}{,bslash,backslash,directory,location,}%
      }%
    }%
}
\newcommand\tw@define@menu@macro[3]
  {%
    \IfNoValueTF{#3}
      {\tw@mk@exp@Nnno\tw@define@menu@macro@{#1}{#2}\tw@default@input@sep}
      {\tw@define@menu@macro@{#1}{#2}{#3}}%
  }
\newcommand\tw@define@menu@macro@[4]
  {%
    \ifcsundef{tw@style@#4@sep}
      {%
        \tw@mk@error
          {%
            Can't define menu macro \string#2\space,\MessageBreak
            because the style '#4' is not available!%
          }%
      }
      {%
        \csdef{tw@parse@menu@list@\tw@mk@string#2}##1%
          {%
```

```
            \def\CurrentMenuElement{##1}%
            \tw@mk@iflastindris
              {%
                \ifnum\tw@mk@indrisnr=\@ne
                  \@nameuse{tw@style@#4@single}%
                \else
                  \@nameuse{tw@style@#4@sep}%
                  \@nameuse{tw@style@#4@last}%
                \fi
              }
              {%
                \ifnum\tw@mk@indrisnr=\@ne
                  \@nameuse{tw@style@#4@first}%
                \else
                  \@nameuse{tw@style@#4@sep}%
                  \@nameuse{tw@style@#4@mid}%
                \fi
              }%
          }%
        #1#2{+O{#3}+m}%
          {%
            \leavevmode
            \begingroup
              \def\tw@current@color@theme
                {\csname tw@style@#4@color@theme\endcsname}%
              \@nameuse{tw@style@#4@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{\tw@mk@trimspaces{##1}}%
                }%
              \begingroup
                \letcs{\tw@mk@tempb}{tw@parse@menu@list@\tw@mk@string#2}%
                \tw@mk@indrisloop\tw@mk@tempa\tw@menu@list\tw@mk@tempb
              \endgroup
              \@nameuse{tw@style@#4@post}%
            \endgroup
          }%
      }%
  }
```

6.7.2 User-level commands

\newmenumacro
\renewmenumacro
\providemenumacro

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

```
\NewDocumentCommand{\newmenumacro}{m o m}{%
   \ifcsundef{\tw@mk@string#1}{%
      \tw@define@menu@macro\NewDocumentCommand{#1}{#2}{#3}%
   }{%
      \tw@mk@error{Menu macro '\string#1' already defined!\MessageBreak
      Use \string\renewmenustyle\space instead.}%
```

```
    }%
}
\NewDocumentCommand{\renewmenumacro}{m o m}{%
    \tw@define@menu@macro\RenewDocumentCommand{#1}{#2}{#3}%
}
\NewDocumentCommand{\providemenumacro}{m o m}{%
    \ifcsundef{\tw@mk@string#1}{%
        \tw@define@menu@macro\ProvideDocumentCommand{#1}{#2}{#3}%
    }{%
        \tw@mk@warning{Menu macro '\string#1' already defined!\MessageBreak
        Use \string\renewmenustyle\space to redefine it.}%
    }%
}
```

6.7.3 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 statement:

```
\fi
```

6.8 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 macros.

`\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%
       #2%
    }%
}
```

`\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 appearance differs between Mac and Windows. To use this macro we assume that the `tw@mk@`⟨*key*⟩ commands are defined.

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

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

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

Next thing to do is to set up some Ti*k*Z-styles.

```
\tikzset{
   menukeys key symbol/.style={
      rounded corners=0pt,
      line width=0.1ex,
      baseline={(0,0)},
   },
   menukeys thick/.style={line width=0.25ex},
}
```

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

`\shift`

```
\normalsize
\tw@make@key@box{shift}{%
   \begin{tikzpicture}[yshift=-0.1ex,menukeys key symbol]
      \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}%
}
\tw@make@key@macro{\shift}
```

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

`\capslock`

```
\tw@make@key@box{capslock@mac}{%
   \begin{tikzpicture}[yshift=-0.1ex,menukeys key symbol]
      \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,menukeys key symbol]
      \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}%
}
\tw@make@key@macro*{\capslock}
```

Here are the other macros:

`\tab`

```
\tw@make@key@box{tab@mac}{%
   \begin{tikzpicture}[yshift=0.6ex,menukeys key symbol]
      \draw [->] (0,0) -- (1em,0);
      \draw (1em,-0.35ex) -- (1em,0.35ex);
   \end{tikzpicture}%
}
\tw@make@key@box{tab@win}{%
   \begin{tikzpicture}[yshift=0.1ex,menukeys key symbol]
      \draw [->] (0.2em,0) -- (1.2em,0);
      \draw (1.2em,-0.35ex) -- (1.2em,0.35ex);
      \draw [<-] (0,1ex) -- (1em,1ex);
      \draw (0,0.65ex) -- (0,1.35ex);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\tab}
```

`\esc`

`\oldesc`

```
\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,menukeys key symbol]
```

```
         \draw [->] (0.5ex,0.5ex) -- ++(135:1.1ex);
         \draw (0.5ex,0.5ex) ++(105:0.6ex) arc (105:-195:0.6ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\esc}
\def\tw@mk@oldesc@win{Esc}
\tw@define@mackey{%
   \def\tw@mk@oldesc@mac{esc}
}{%
   \tw@make@key@box{oldesc@mac}{%
      \begin{tikzpicture}[yshift=-0.1ex,menukeys key symbol]
         \draw [->] (0.5ex,0.5ex) -- ++(45:1.1ex);
         \draw (0.5ex,0.5ex) ++(15:0.6ex) arc (15:-285:0.6ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\oldesc}
```

\ctrl

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

\Alt
\AltGr

```
\def\tw@mk@Alt@win{Alt}
\tw@define@mackey{%
   \def\tw@mk@Alt@mac{alt}%
}{%
   \tw@make@key@box{Alt@mac}{%
      \begin{tikzpicture}[yshift=-0.1ex,menukeys key symbol]
         \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}
```

\cmd

```
\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,menukeys key symbol]
         \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}
```

\Space
\SPACE

```
\providecommand*{\Space}{\expandonce{\rule{3em}{0pt}}}
\newcommand{\spacename}{Space}
\providecommand*{\SPACE}{\expandonce{\rule{2em}{0pt}\spacename\rule{2em}{0pt}}}
```

\return

```
\tw@make@key@box{return@mac}{%
   \begin{tikzpicture}[yshift=0.25ex,menukeys key symbol]
      \draw [->, rounded corners=0.2ex] (1.25ex,1ex) -| %
            (2ex,0) -- (0,0);
   \end{tikzpicture}%
}
\tw@make@key@box{return@win}{%
   \begin{tikzpicture}[menukeys key symbol]
      \draw [->] (1ex,1.25ex) |- (0,0);
   \end{tikzpicture}%
}
\tw@make@key@macro*{\return}
```

\enter

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

\winmenu

```
\def\tw@mk@winmenu@mac{%
   \tw@mk@warning{'\string\winmenu' only for Windows!}%
}
\tw@make@key@box{winmenu@win}{%
   \begin{tikzpicture}[yshift=-0.2ex,menukeys key symbol]
      \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}
```

\backspace

```
\tw@make@key@box{backspace}{%
   \begin{tikzpicture}[yshift=0.65ex,menukeys key symbol]
      \draw [<-,menukeys thick] (0,0) -- (1.35em,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}[yshift=0.2ex,menukeys key symbol]
         \draw (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}
\def\tw@mk@backdel@win{\delname}
\tw@define@mackey{%
   \def\tw@mk@backdel@mac{\delname}%
}{%
   \tw@make@key@box{backdel@mac}{%
      \begin{tikzpicture}[yshift=0.2ex,menukeys key symbol]
         \draw (2ex,0) -- (0.5ex,0) -- (0,0.5ex) --%
               (0.5ex,1ex) -- (2ex,1ex) -- cycle;
         \draw (1ex,0.2ex) -- (1.6ex,0.8ex);
         \draw (1ex,0.8ex) -- (1.6ex,0.2ex);
      \end{tikzpicture}%
   }%
}
\tw@make@key@macro*{\backdel}
```

Lastly we define the arrow macros:

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

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

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

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

```
   \begin{tikzpicture}[yshift=0.5ex,menukeys key symbol]
      \draw [->] (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
```

⟨/pkg⟩

7 Change history

8 Macro index

Numbers written in bold face refer to the page where the corresponding entry is described; italic numbers refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

M

mackeys (option) **4**, **13**
\menu **4**, *723*
menus (style) **5**

N

\newmenucolortheme **11**, *120*, 167, 168
\newmenumacro **12**, *703*, 723, 724, 725
\newmenustyle **9**, *234*, 269
\newmenustylesimple **9**, *234*, 234

O

\oldesc *816*
Options:
  definekeys **4**
  definemenumacros **4**
  mackeys **4**, **13**
  os **4**
os (option) **4**

P

\PassOptionsToPackage 110
paths (style) **7**
pathswithblackfolder (style) **7**
pathswithfolder (style) **7**
\prg 14
\protected 634
\providemenumacro **12**, *703*
\providemenustyle **10**, *234*, 289
\providemenustylesimple **10**, *234*, 254

R

\relsize 354, 385, 415, 446, 454, 466, 479, 493, 501, 512, 524, 536, 551
\renewmenucolortheme **12**, *120*, 149
\renewmenumacro **12**, *703*
\renewmenustyle **10**, *234*, 278, 282, 299, 708, 719
\renewmenustylesimple **10**, *234*, 243, 247, 264
\return *876*
roundedkeys (style) **6**
roundedmenus (style) **5**

S

\seq 21, 23, 27, 28, 31, 35, 39, 40, 41, 45, 49, 55, 60, 62, 65
shadowedangularkeys (style) **6**
shadowedroundedkeys (style) **6**
\shift *776*
\SPACE *873*
\Space *873*
\spacename **13**, 874, 875
Styles:
  angularkeys **6**
  angularmenus **5**
  hyphenatepathswithblackfolder **8**
  hyphenatepathswithfolder **8**
  hyphenatepaths **8**
  menus **5**
  pathswithblackfolder **7**
  pathswithfolder **7**
  paths **7**
  roundedkeys **6**
  roundedmenus **5**
  shadowedangularkeys **6**
  shadowedroundedkeys **6**
  typewriterkeys **7**

T

\tab *801*
\tikzset 341, 346, 377, 408, 438, 458, 486, 505, 531, 768
\tl 11, 14, 17, 25
\tw@current@color@theme 165, 682
\tw@current@style 204, 205, 210
\tw@declare@style *201*, 272, 274, 284, 286, 292, 294, 359, 390, 420
\tw@declare@style@extra@args *201*
\tw@declare@style@simple *174*, 237, 239, 249, 251, 257, 259, 448, 472, 495, 518, 538, 555, 569
\tw@default@input@sep *627*, 643
\tw@default@post *169*, 175, 202
\tw@default@pre *169*, 175, 208
\tw@default@sep *169*, 175, 208
\tw@define@mackey *763*, 817, 829, 845, 860, 916, 930
\tw@define@menu@macro *629*, 705, 712, 716
\tw@define@menu@macro@ *629*
\tw@make@color@theme *112*, 122, 129
\tw@make@key@box *728*, 777, 785, 793, 801, 807, 820, 832, 848, 863, 876, 882, 889, 899, 908, 919, 933, 943, 950, 957, 964
\tw@make@key@macro *738*, 784, 800, 815, 827, 839, 843, 855, 872, 887, 895, 907, 913, 928, 942, 948, 955, 962, 969
\tw@menu@list 687, 691, 696
\tw@mk@Alt@mac 846
\tw@mk@Alt@win 844
\tw@mk@backdel@mac 931
\tw@mk@backdel@win 929
\tw@mk@cmd@mac 861
\tw@mk@cmd@win 857
\tw@mk@ctrl@mac 842
\tw@mk@ctrl@win 841

U

W