Tobias Weh
mail@tobiw.de
http://tobiw.de/en
http://github.com/tweh/menukeys
http://www.ctan.org/pkg/menukeys

2016/08/08 — v1.5

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 list parsing code at http://tex.stackexchange.com/a/44989/4918.
## Contents

1 Introduction 3
2 Installation 3
3 Package loading and options 4
4 Usage 4
   4.1 Basics ......................................................... 4
   4.2 Styles ......................................................... 5
      4.2.1 Predefined styles ................................. 5
      4.2.2 Declaring styles ............................... 9
      4.2.3 Copying styles .............................. 10
      4.2.4 Changing styles .......................... 10
   4.3 Color themes .............................................. 11
      4.3.1 Predefined themes .......................... 11
      4.3.2 Create a theme .................................. 11
      4.3.3 Copy a theme .................................. 12
      4.3.4 Change a theme ................................ 12
   4.4 Menu macros .............................................. 12
      4.4.1 Predefined menu macros ................. 12
      4.4.2 Defining or changing menu macros .... 12
   4.5 Keys ......................................................... 13
5 Known issues and bugs 14
6 Implementation 14
   6.1 Required packages ........................................ 14
   6.2 Helper macros .......................................... 15
   6.3 Options ..................................................... 15
   6.4 Workarounds .............................................. 16
      6.4.1 `hyperref`'s `colorlinks` option ................. 16
   6.5 Color themes .............................................. 16
      6.5.1 Internal commands ............................... 16
      6.5.2 User-level commands .......................... 16
      6.5.3 Predefined themes .......................... 18
   6.6 Styles ......................................................... 18
      6.6.1 Internal commands ............................... 18
      6.6.2 User-level commands .......................... 19
      6.6.3 Copying and changing .......................... 21
      6.6.4 Predefined styles .......................... 22
   6.7 Menu macros .............................................. 28
      6.7.1 Internal commands ............................... 28
      6.7.2 User-level commands .......................... 29
      6.7.3 Predefined menu macros ....... 29
   6.8 Keys ......................................................... 30
7 Change history  
8 Macro index
1 Introduction

The \texttt{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\textit{k}Z\textsuperscript{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

\begin{verbatim}
To set the unit of the rulers go to \texttt{\textbackslash menu\{Extras > Settings > Rulers\}}
and choose between millimeters, inches and pixels. The shortcut to view the rulers is \texttt{\textbackslash keys\{cmd + R\}}. Pressing these keys again will hide the rulers.

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

and get this:

To set the unit of the rulers go to \texttt{Extra\{Extra\{Extra\} \textbackslash Settings \textbackslash Rulers\}} and choose between millimeters, inches and pixels. The shortcut to view the rulers is \texttt{\textbackslash cmd + R}. Pressing these keys again will hide the rulers.

The standard path for saving your document is \texttt{Macintosh HD/Users/Your Name/Documents} but you can change it at \texttt{Extra\{Extra\{Extra\} \textbackslash Settings \textbackslash Saving\}} by clicking \texttt{Change save path}.

The package is loaded as usual via

\texttt{\textbackslash usepackage\{menukeys\}}

2 Installation

To install \texttt{menukeys} manually run

\texttt{latex menukeys.ins}

and copy \texttt{menukeys.sty} to a path where \texttt{\LaTeX} can find it.

To typeset this manual run

\texttt{pdflatex menukeys.dtx}
\texttt{makeindex -s gglo.ist -o menukeys.gls menukeys.glo}
\texttt{makeindex -s gind.ist -o menukeys.ind menukeys.idx}
\texttt{pdflatex menukeys.dtx}
\texttt{pdflatex menukeys.dtx}

\textsuperscript{1} See \texttt{http://www.ctan.org/pkg/pgf}.

\footnotetext{1}{See \texttt{http://www.ctan.org/pkg/pgf}.}
3 Package loading and options

Since menukeys uses catoptions, which does some heavy changes on key-value options, it is recommended to load menukeys as the last package (even after hyperref)!

These are the possible options:

**definemenumacros**: Most of menukeys’ macros should not conflict with other packages but the predefined menu macros should be short and easy-to-read commands, which means that \texttt{\textbackslash{}menu(A,B,C)} is preferred against \texttt{\textbackslash{}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**: Equal to definemenumacros for the key macros. The default value is true.

**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**: 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**: Use this if you want hyperref’s colored links, since you can’t use the hyperref option colorlinks directly (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 \texttt{\textbackslash{}menu\{\texttt{(menu sequence)}\}}, with > as default input list separator, \texttt{\textbackslash{}directory\{\texttt{(path and files)}\}} with / as default separator and \texttt{\textbackslash{}keys\{\texttt{(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 \texttt{\textbackslash{}menu[,]\{\texttt{(menu sequence)}\}}.\footnote{See \url{http://tex.stackexchange.com/q/237683/4918} and \url{https://github.com/tweh/menukeys/issues/41}.}

The possible input separators are /, =, *, +, ,, ;, ;, -, >, < and bslash (to use \ as separator). You can hide a separator from the parser by putting a separator globally it’s recommended to renew the menu macro as described in section 4.4.

\footnote{If you want to change the input separator, globally it’s recommended to renew the menu macro as described in section 4.4.}
part of the sequence in braces. Spaces around the separator will be ignored, i.e. \keys{\text{ctrl}+C} equals \keys{\text{ctrl} + C}.

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

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

<table>
<thead>
<tr>
<th>Name: menus</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Extras Preferences</td>
</tr>
<tr>
<td>Menu</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: roundedmenus</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Extras Preferences</td>
</tr>
<tr>
<td>Menu</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: angularmenus</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Extras Preferences</td>
</tr>
<tr>
<td>Menu</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>
Name: roundedkeys

[Ctrl] + [Alt] + [Q]

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This \([\text{Ctrl}] + [\text{Alt}] + [\text{Q}]\) is the result of a style which name is \textit{roundedkeys}. And again some blind text without any sense.

\textit{The color of + is taken from optional color B.}

Name: shadowedroundedkeys

[Ctrl] + [Alt] + [Q]

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This \([\text{Ctrl}] + [\text{Alt}] + [\text{Q}]\) is the result of a style which name is \textit{shadowedroundedkeys}. And again some blind text without any sense.

\textit{The color of + is taken from optional color B.}
\textit{The shadow color is taken from optional color C.}

Name: angularkeys

[Ctrl] + [Alt] + [Q]

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This \([\text{Ctrl}] + [\text{Alt}] + [\text{Q}]\) is the result of a style which name is \textit{angularkeys}. And again some blind text without any sense.

\textit{The color of + is taken from optional color B.}

Name: shadowedangularkeys

[Ctrl] + [Alt] + [Q]

S

This is some more or less blind text, to demonstrate how the sequence looks in text. This \([\text{Ctrl}] + [\text{Alt}] + [\text{Q}]\) is the result of a style which name is \textit{shadowedangularkeys}. And again some blind text without any sense.

\textit{The color of + is taken from optional color B.}
\textit{The shadow color is taken from optional color C.}
This is some more or less blind text, to demonstrate how the sequence looks in text. This \(\text{\texttt{Q}}\) is the result of a style which name is \texttt{typewriterkeys}. And again some blind text without any sense.

\textit{The color of \texttt{Q} is taken from optional color B.}

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

\textit{The sep color is taken from optional color C.}

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

\textit{The folder draw color is taken from optional color B.}
\textit{The folder fill color is taken from optional color A.}
\textit{The sep color is taken from optional color C.}

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

\textit{The folder draw color is taken from optional color B.}
\textit{The folder fill color is taken from optional color C.}
\textit{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\ALongUserNameHere\ALongerFolderPath\MyFile.tex

This is some more or less blind text, to demonstrate how the sequence looks in text. This C:\Database\User\ALongUserNameHere\ALongerFolderPath\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.*

Name: hyphenatepathswithfolder
\C: / Database / User / ALongUserNameHere / ALongerFolderPath / MyFile.tex

This is some more or less blind text, to demonstrate how the sequence looks in text. This \C: / Database / User / ALongUserNameHere / ALongerFolderPath / 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.*

Name: hyphenatepathswithblackfolder
\C: / Database / User / ALongUserNameHere / ALongerFolderPath / 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 / ALongerFolderPath / 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.*
\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\kZ-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\kZ-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\kZ style.
- **last** is the style for the last list element. It has to be a Ti\kZ style.
- **single** this style is used if the list contains only one element. It has to be a Ti\kZ 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 TikZ 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. To access the current list element use \CurrentMenuElement.

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

\newmenustylesimple*{mystyle}{$\ast$}\%{\textbf{\CurrentMenuElement}}

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{mystyle}{mycopy}

4.2.4 Changing styles

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}
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 \texttt{node} background, a \texttt{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: \(\square\) Border: \(\square\) Text: \(\square\) (A: \(\square\) B: \(\square\) C: \(\square\))

Name: blacknwhite
Background: \(\square\) Border: \(\square\) Text: \(\square\) (A: \(\square\) B: \(\square\) C: \(\square\))

4.3.2 Create a theme

To create a new theme use \texttt{\newmenucolortheme}. It uses the following arguments:

\texttt{\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 \texttt{node} background.

br  is the color definition for the \texttt{node} border.

txt is the color definition for the \texttt{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

\texttt{\newmenucolortheme{mycolors}{named}{red}{green}{blue}}
4.3.3 Copy a theme

\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 macros” 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 will 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}{b\textbackslash}{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.

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.

\textbf{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 \(\key\text{mac}\) and \(\key\text{win}\) that will always give the right symbol.

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

<table>
<thead>
<tr>
<th>Macro</th>
<th>Mac</th>
<th>Win.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\shift</td>
<td>′</td>
<td>′</td>
</tr>
<tr>
<td>\capslock</td>
<td>ㅎ</td>
<td>ㅎ</td>
</tr>
<tr>
<td>\tab</td>
<td>⏯</td>
<td>⏯</td>
</tr>
<tr>
<td>\esc</td>
<td>esc / ⏯</td>
<td>Esc</td>
</tr>
<tr>
<td>\oldesc</td>
<td>esc / ⏯</td>
<td>Esc</td>
</tr>
<tr>
<td>\ctrl</td>
<td>ctrl</td>
<td>Ctrl</td>
</tr>
<tr>
<td>\Alt</td>
<td>alt / ⏯</td>
<td>Alt</td>
</tr>
<tr>
<td>\AltGr</td>
<td>Alt Gr</td>
<td>Alt Gr</td>
</tr>
<tr>
<td>\cmd</td>
<td>cmd / ⏯</td>
<td></td>
</tr>
<tr>
<td>\Space</td>
<td>[empty sp.]</td>
<td>[empty sp.]</td>
</tr>
<tr>
<td>\SPACE</td>
<td>Space</td>
<td>Space</td>
</tr>
<tr>
<td>\return</td>
<td>′</td>
<td>′</td>
</tr>
<tr>
<td>\enter</td>
<td>↵</td>
<td>↵</td>
</tr>
</tbody>
</table>

The macro \arrowkey{\(\text{direction}\)} is a little special since it takes the direction as a single character ′, v (lower case v), > or <.

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

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`.

  Example

  Set `xcolor` to cmyk model:

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

- Using `hyperref` with the `colorlinks` options causes an option clash. If you want colored links please load `hyperref` without this option and load `menukeys` with `hyperrefcolorlinks`.

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

6.1 Required packages

Load the required packages

1 \RequiresPackage{xparse}
2 \RequiresPackage{xstring}
3 \RequiresPackage{etoolbox}

Furthermore we need TikZ and some of its libraries,

4 \RequiresPackage{tikz}
5 \usetikzlibrary{calc,shapes.symbols,shadows}

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

6 \RequiresPackage{xcolor}
7 \RequiresPackage{adjustbox}

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

8 \RequiresPackage{relsize}

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

9 \RequiresPackage{catoptions}[2011/12/07]
6.2 Helper macros

Define macros to call \PackageError and warnings

\newcommand*{\tw@mk@error}[2][]{\PackageError{menukeys}{#2}{#1}}%
\newcommand*{\tw@mk@warning}[1][]{\PackageWarning{menukeys}{#1}}%
\newcommand*{\tw@mk@warning@noline}[1][]{\PackageWarningNoLine{menukeys}{#1}}%

Some commands for temporary use:
\def\tw@mk@tempa{}
\def\tw@mk@tempb{}

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 “slove” some incompatibilities:

6.4.1 hyperref’s colorlinks option

Since the colorlinks option of hyperref loads color (with some kind of \AtBeginDocument it results in an option clas due to the changes made by catoptions. Thus one can’t use colorlinks. Here we provide the code to activate colored links without the extra loading of color.

\iftw@mk@hyperref\colorlinks
\Hy@AtBeginDocument{% (hyperref.sty, line 4790)
\def@pdfborder{0 0 0}% (hyperref.sty, line 4806...)
\let@pdfborderstyle@empty
\ifHy@typexml% <--------------+
\else% | This part
\Hy@CatcodeWrapper{% | bust be
\RequirePackage{color}% | omitted
\fi% <------------------------+
\def\Hy@colorlink#1{%
\begingroup
\HyColor@UseColor#1%
\Hy@endcolorlink%
\Hy@Info{Link coloring ON}%
}\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
\renewmenucolortheme
After that we define the user-level commands:
\NewDocumentCommand{\newmenucolortheme}{ m m m m m O{#3} O{#4} O{#5} }{%
\ifundefinedcolor{tw@color@theme@#1@bg}{%#2}{#3}%
\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}%
}%
\tw@make@color@theme{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}%
Lastly we define the changing and copying commands
\newcommand*{\changemenucolor}[4]{%
  \IfSubStr{ bg br txt }{ #2 }{% 
    \definecolor{tw@color@theme@#1@#2}{#3}{#4}%
  }{% 
    \tw@mk@error{No such color element ('#2')!
      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!
      Use \string\renewmenucolortheme\space instead.}%
  }%}

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!
      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}%
  }%}

To use a color of a theme we define \usemenucolor as following.
\newcommand{\usemenucolor}[1]{%
  tw@color@theme@tw@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][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⟩.

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.

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};}}
}
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 
}%

6.6.2 User-level commands

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

\NewDocumentCommand{\newmenustylesimple}{s m}{% 
\ifcsundefined{tw@style@#2@pre}{% 
\tw@declare@style@simple*{#2} 
}% 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

\NewDocumentCommand{\renewmenustylesimple}{s m}{% 
\tw@declare@style@simple@*{#2} 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

\NewDocumentCommand{\providemenustylesimple}{s m}{% 
\tw@declare@style@simple@*{#2} 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

\NewDocumentCommand{\newmenustyle}{s m}{% 
\ifcsundefined{tw@style@#2@pre}{% 
\tw@declare@style@*{#2} 
}% 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

\NewDocumentCommand{\renewmenustyle}{s m}{% 
\tw@declare@style@*{#2} 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

\NewDocumentCommand{\providemenustyle}{s m}{% 
\tw@declare@style@*{#2} 
\csdef{tw@current@style@post}{#1} 
\csdef{tw@current@style@color@theme}{#2} 
}%

!!indent!!newmenustylesimple!!indent!!renewmenustylesimple!!indent!!providemenustylesimple

!!indent!!newmenustyle!!indent!!renewmenustyle!!indent!!providemenustyle
\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 o m}\
}%\

\NewDocumentCommand\renewmenustyle{s m}{%\
\IfBooleanTF{#1}{%\
\tw@declare@style*{#2}\
}%\
\tw@declare@style{#2}\
}%
\NewDocumentCommand{\providemenustyle}{s m}{% 
\ifsundef{tw@style@#2@pre}{% 
  \IfBooleanTF{#1}{% 
    \tw@declare@style*{#2}% 
    \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 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]{% 
\ifsundef{tw@style@#1@pre}{% 
  \ifsundef{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.
\newcommand{\changemenuelement}{s m m m}{% 
\ifsundef{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} %
    }{% 
      \csdef{tw@style@#2@#3}{#4} %
    }% 
  }% 
}%

22
6.6.4 Predefined styles

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

tw@set@tikz@colors

First we define a TiKZ-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 TiKZ-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,
}}

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};}
  }

\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};}
\]
\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{\node[\tw@angularkeys@base]{\strut\color{\usemenucolor{txt}}\CurrentMenuElement};} 
}\{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, 
  general shadow={
    shadow xshift=.2ex, shadow yshift=-.15ex, 
    fill=\usemenucolor{c}, 
  }, 
}}
\tw@declare@style@simple*{shadowedangularkeys}{
  \tikz{\node[\tw@shadowedangularkeys@base]{\strut\color{\usemenucolor{txt}}\CurrentMenuElement};} 
}\{gray\}
6.7 Menu macros

6.7.1 Internal commands

\tw@default@input@sep

First we define our default input separator

\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{\,}{\|}{\,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!}%
}
\else
\csdef{tw@parse@menu@list@\expandafter\@gobble\string#1}##1{%
\iflastindris
\ifnum\indrisnr=\@ne
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@single}%
\else
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@last}%
\fi
\else
\ifnum\indrisnr=\@ne
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@first}%
\else
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@mid}%
\fi
\fi
\fi
\csdef{tw@parse@menu@list@\expandafter\@gobble\string#1}##1{%
\iflastindris
\ifnum\indrisnr=\@ne
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@single}%
\else
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@last}%
\fi
\else
\ifnum\indrisnr=\@one
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@first}%
\else
\def\CurrentMenuElement{##1}%
\@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@mid}%
\fi
\fi
\expandafter\newcommand\csname\expandafter\@gobble\string#1\endcsname[2][2]{%
\leavevmode%
{\def\tw@current@color@theme{\csname tw@style@#3@color@theme\endcsname}%
\@nameuse{tw@style@#3@pre}%
\expandafter\newcommand\csname\expandafter\@gobble\string#1\endcsname[2][2]{%
Now it’s time to build the user-level commands

6.7.2 User-level commands

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

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:

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\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#2\endcsname{%
      \expandonce{\maxsizebox{!}{1.8ex}{%\@nameuse{tw@mk@\expandafter\@gobble\string#2}}%
    %}
  }%}
  \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}}%
  %}
}}
The last helping macro is \texttt{\tw@define\mackey}. We use it to execute code depending on the \texttt{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\textsc{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 are prepared to generate the key macros. I will be nearly the same way for all keys. Step one is to build a \texttt{\tw@mk\langle key\rangle} macro and then we define the user-level command \texttt{\langle key\rangle}.

\texttt{\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 \texttt{\tw@mk\langle key\rangle@mac} and \texttt{\tw@mk\langle key\rangle@win}. And then use the starred version \texttt{\tw@make\key@macro*} which creates \texttt{\langle key\rangle} that depends on the \texttt{os} option, \texttt{\langle key\rangle@mac} and \texttt{\langle key\rangle@win}, that are not affected by \texttt{os}.

\texttt{\capslock}\normalsize

\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);
Here are the other macros:

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

```latex
\tw@make@key@macro*{\tab}
```

```latex
\tw@make@key@macro*{\esc}
```

```latex
\tw@make@key@macro*{\oldesc}
```
\draw (0.5ex,0.5ex) ++(15:0.6ex) arc (15:-285:0.6ex);
\end{tikzpicture}
}\}
\tw@make@key@macro*{\oldesc}
\ctrl
\providecommand{\ctrlname}{\texttt{Ctrl}}
\def\tw@mk@ctrl@win{\ctrlname}
\def\tw@mk@ctrl@mac{\texttt{ctrl}}
\tw@make@key@macro*{\ctrl}
\Alt
\providecommand*{\AltGr}{\texttt{Alt\,Gr}}
\cmd
\providecommand*{\Space}{\expandonce{\rule{3em}{0pt}}}
\newcommand*{\spacename}{\texttt{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=0.5ex,menukeys key symbol]
  \draw [<-,menukeys thick] (0,0) -- (1.25em,0);
\end{tikzpicture}%
\tw@make@key@macro{\backspace}
\del
\backdel
\providecommand{\delname}{Del.}
\def\tw@mk@del@win{\delname}
Lastly we define the arrow macros:

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

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

\begin{tikzpicture}[yshift=0.5ex,menukeys key symbol]
\draw [->] (0,0) -- (0.8em,0);
\end{tikzpicture}
\tw@make@key@macro{\arrowkeyright}
And the `\arrowkey` macro that get’s it’s direction as argument.

```latex
\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`
7 Change history

v1.0
General: Initial version ........ 1

v1.1
\directory: Renamed \path to
\directory because it crashes
with \biblatex .......... 29
General: Improved manual .... 1
Load xcolor before \menukeys. 14

v1.1a
\newmenumacro: Added a line to
make a new macro robust. ..... 29
\tw@define\menumacro: Fixed
minor bug, that causes a
warning about robustifying
(issue #23), by deleting the
line to make the command
robust. ...................... 28

v1.2
\tw@define\menumacro: Added
\leavevmode ................ 28
Replaced \edef by
\protected@edef ............ 28
General: Added \normalsize
before symbol definitions to
make the \ex unit available .... 1
Added \SPACE and \spacename . 1
Fixed GitHub issues #9, #10,
#11, #13, #17, #24 and #26 . 1

v1.2a
Tidy up version and date .... 1

v1.2c
\tw@define\menumacro: Replaced
\protected@edef by \def ... 28

v1.3
General: Added TikZ-styles for the
key symbols. .............. 1
Improved key symbols. .... 1

v1.4
\backdel: Added \backdel .... 34
\oldesc: Fixed direction of
\escmac; added \oldesc .... 32
General: Extended color theme
features. .................. 1
The \path... styles now use the
text color of the selected color
theme (fix issue #16). .... 1

v1.5
General: New option
\hyperref\colorlinks .... 16

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.

Symbols
\pdfborder ............... 46
\pdfborderstyle ........ 47
\tw@hyphenatepaths@Warnigfalse . 524
\tw@hyphenatepaths@Warnigtrue . 512
\arrowkey .............. 13, 891
\arrowkeydown .......... 864, 893
\arrowkeyleft .......... 864, 894
\arrowkeyright ........ 864, 895
\arrowkeyup .......... 864, 892
\Alt .................. 765
\AltGr ................ 765

angularkeys (style) ............ 6
angularmenus (style) ........ 5
| \backdel                      | 835 |
| \backspace                    | 829 |
| blacknwhite (theme)           | 11  |
| \capslock                     | 706 |
| \changemenucolor             | 12 , 81 |
| \changemenucolortheme        | 10 , 102 |
| \changemenuelement           | 10 , 271, 549, 555, 561, 567 |
| \color                       | 778 |
| \cmd                          | 284, 312, 317, 321, 324, 343, 348, 352, 355, 373, 378, 382, 385, 401, 425, 445, 468, 489, 491, 500, 528 |
| Color themes:                |     |
| blacknwhite                  | 11  |
| gray                         | 11  |
| \copymenucolortheme         | 12 , 81 |
| \copymenustyle               | 10 , 253, 548, 554, 560, 566 |
| \ctrl                        | 761 |
| \ctrlname                    | 13 , 761, 762 |
| \CurrentMenuElement          | 10 , 139, 142, 145, 148, 171, 174, 177, 180, 284, 312, 317, 321, 324, 343, 348, 352, 355, 373, 378, 382, 385, 401, 425, 445, 468, 484, 500, 529, 572, 590, 593, 598, 601 |
| \definekeys (option)         | 4   |
| definemenumacros (option)    | 4   |
| \del                          | 835 |
| \delname                      | 13 , 835, 836, 838, 850, 852 |
| \directory                    | 4 , 644 |
| \drawtikzf folder            | 9 , 539, 550, 556, 562, 568 |
| \enter                       | 809 |
| \esc                          | 737 |
| \font                         | 527 |
| gray (theme)                  | 11  |
| \Hy@AtBeginDocument          | 45  |
| \Hy@CatcodeWrapper           | 50  |
| \Hy@colorlink                | 54  |
| \Hy@endcolorlink             | 58  |
| \Hy@Info                      | 59  |
| \Hy@Color@UseColor           | 56  |
| \hyperrefcolorlinks (option) | 4   |
| \hyphenatepaths (style)      | 8   |
| \hyphenatepathswithblackfolder (style) | 8 |
| \hyphenatepathswithfolder (style) | 8 |
| \ifttw@hyphenatepaths@warning | 511 , 521 |
| \ifHy@xml                     | 48  |
| \iftw@mk@definekeys          | 648 |
| \iftw@mk@definemenumacros    | 643 |
| \iftw@mk@hyperrefcolorlinks  | 44  |
| \keys                         | 4 , 644 |
| \mackeys (option)             | 4 , 13 |
| \menu                         | 4 , 644 |
| \menus (style)               | 5   |
| \newmenucolortheme           | 11 , 70, 117, 118 |
| \newmenumacro                | 12 , 622, 644, 645, 646 |
| \newmenustyle                | 9 , 184, 219 |
| \newmenustylesimple          | 9 , 184, 184 |
| \providemenumacro            | 13 , 622 |
| \providemenustyle            | 11 , 184, 239 |
| \providemenustylesimple      | 11 , 184, 204 |
| Options:                     |     |
| definekeys                   | 4   |
| definemenumacros             | 4   |
| \hyperrefcolorlinks          | 4   |
| mackeys                      | 4 , 13 |
| os                           | 4   |
| os (option)                  | 4   |
| \paths (style)               | 7   |
| \pathswithblackfolder (style) | 7 |
| \pathswithfolder (style)     | 7   |
| \providemenumacro            | 13 , 622 |
| \providemenustyle            | 11 , 184, 239 |
| \providemenustylesimple      | 11 , 184, 204 |
R
\relsize 304, 335, 365, 396, 404, 416, 429, 440, 448, 459, 471, 480, 495
\renewmenucolortheme 12, 70, 99
\renewnumacro 13, 622
\renewmenustyle 11, 184, 228, 232, 249, 628, 640
\renewmenustylesimple 11, 184, 193, 197, 214
\return 797
roundedkeys (style) 6
roundedmenus (style) 5

S
shadowedangularkeys (style) 6
shadowedroundedkeys (style) 6
\shift 697
\SPACE 794
\Space 794
\spacename 13, 795, 796

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 722
\tikzset 291, 296, 327, 358, 388, 408, 433, 452, 475, 689
\tw@current@color@theme 115, 608
\tw@current@style 154, 155, 160
\tw@declare@style 157, 222, 224, 234, 236, 242, 244, 309, 340, 370
\tw@declare@style@simple 124, 187, 189, 199, 201, 207, 209, 398, 422, 442, 465, 482, 499, 513
\tw@declare@style@extrargs 151
\tw@declare@style@extra@args 151
\tw@default@input@sep 571, 581, 622, 631, 635
\tw@default@post 119, 125, 152
\tw@default@pre 119, 125, 158
\tw@default@sep 119, 125, 158
\tw@define@mackey 684, 738, 750, 766, 781, 837, 851
\tw@define@menu@macro 573, 624, 633, 637
\tw@define@mackey@box 649, 698, 706, 714, 722, 728, 741, 753, 769, 784, 797, 803, 810, 820, 829, 840, 854, 864, 871, 878, 885
\tw@define@key@box 611, 613, 616
\tw@define@mackey@mac 762, 763, 765, 767, 782, 788, 809, 850
\tw@define@closel@mac 852
\tw@define@closel@win 850
\tw@define@closel@win 778
\tw@define@closel@win 763
\tw@define@dels@mac 836
\tw@define@dels@mac 838
\tw@define@dels@win 836
\tw@define@dels@win 838
\tw@define@enter@win 809
\tw@define@error 10, 37, 41, 74, 85, 98, 104, 108, 192, 227, 256, 268, 273, 287, 584, 627, 896
\tw@define@esc@mac 739
\tw@define@esc@win 737
\tw@define@gobble@args 21, 194, 215, 229, 250
\tw@define@mackeys 40, 685, 686
\tw@define@desc@mac 751
\tw@define@desc@mac 749
\tw@define@os 36, 663
\tw@define@tempa 19, 22, 23, 611, 631, 616
\tw@define@tempb 19, 615, 616
\tw@define@test@input@sep 576, 610
\tw@define@warning 10, 212, 247, 522, 639, 779, 818
\tw@define@warning@oneline 10
\tw@define@menu@mac 817
\tw@define@menuname@mac 817
\tw@define@set@tikz@colors 291
\tw@typewriterkeys@curr@elem ... 373, 378, 382, 385, 401, 404.
typewriterkeys (style) ............ 7 468, 471, 489, 491, 495, 500,
                        504, 528, 533, 550, 556, 562, 568
                      
U                      
\usemenucolor ............ 10, 114,
                         284, 292, 293, 294, 312, 317,
                         321, 324, 343, 348, 352, 355,  817
WIN                     
\winmenu .................... 817