

# MENUKEYS

Tobias Weh

[mail@tobias-weh.de](mailto:mail@tobias-weh.de)

<http://www.tobias-weh.de>

<http://www.ctan.org/pkg/menukeys>

☞ macros › latex › contrib › menukeys

2012/02/27 — v1.1

## Abstract

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

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

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Installation</b>	<b>4</b>
<b>3</b>	<b>Package options</b>	<b>5</b>
<b>4</b>	<b>Usage</b>	<b>5</b>
4.1	Basics . . . . .	5
4.2	Styles . . . . .	6
4.2.1	Predefined styles . . . . .	6
4.2.2	Declaring styles . . . . .	8
4.2.3	Copying styles . . . . .	10
4.2.4	Changing styles . . . . .	10
4.3	Color themes . . . . .	10
4.3.1	Predefined themes . . . . .	10
4.3.2	Create a theme . . . . .	11
4.3.3	Copy a theme . . . . .	11
4.3.4	Change a theme . . . . .	11
4.4	Menu macros . . . . .	11
4.4.1	Predefined menu macros . . . . .	11
4.4.2	Defining or changing menu macros . . . . .	12
4.5	Keys . . . . .	12
<b>5</b>	<b>Known issues and bugs</b>	<b>13</b>
<b>6</b>	<b>Implementation</b>	<b>14</b>
6.1	Required packages . . . . .	14
6.2	Helper macros . . . . .	14
6.3	Options . . . . .	15
6.4	Color themes . . . . .	15
6.4.1	Internal commands . . . . .	15
6.4.2	User-level commands . . . . .	15
6.4.3	Predefined themes . . . . .	16
6.5	Styles . . . . .	16
6.5.1	Internal commands . . . . .	17
6.5.2	User-level commands . . . . .	18
6.5.3	Copying and changing . . . . .	20
6.5.4	Predefined styles . . . . .	21
6.6	Menu macros . . . . .	26
6.6.1	Internal commands . . . . .	26
6.6.2	User-level commands . . . . .	27
6.6.3	Predefined menu macros . . . . .	28
6.7	Keys . . . . .	28
<b>7</b>	<b>Change history</b>	<b>35</b>



# 1 Introduction

The `menukeys` package is mainly designed to parse and print sequences of software menus, folders and files or keystrokes. The most predefined styles use the power of `TikZ`<sup>1</sup> to format the output

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

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

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

and get this:

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

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

The package is loaded as usual via

```
\usepackage{menukeys}
```

# 2 Installation

To install `menukeys` manually run

```
latex menukeys.ins
```

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

To typeset this manual run

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

---

<sup>1</sup> See <http://www.ctan.org/pkg/pgf>.

### 3 Package options

These are the possible options:

<code>definenummacros</code> (opt.)	<p><b>definenummacros:</b> Most of <code>menukeys</code>' macros should not conflict with other packages<sup>2</sup> but the predefined menu macros should be short and easy to read commands, which means that <code>\menu{A,B,C}</code> is preferred against <code>\printmenusequence{A,B,C}</code>. For that it's not unlikely that these conflict with other packages. To prevent this you can tell <code>menukeys</code> to not define them by calling the option <code>definenummacros=false</code>. The default value is <code>true</code>.</p> <p>If you do so you have to define your own menu macros, see section 4.4 for details.</p>
<code>definekeys</code> (opt.)	<p><b>definekeys:</b> Equal to <code>definenummacros</code> for the key macros. The default value is <code>true</code>.</p>
<code>mackeys</code> (opt.)	<p><b>mackeys:</b> This option allows you to decide whether the mac keys are shown as text (<code>mackeys=text</code>) or symbols (<code>mackeys=symbols</code>). The default value is <code>symbols</code></p>
<code>os</code> (opt.)	<p><b>os:</b> You can specify the OS by saying <code>os=mac</code> or <code>os=win</code>. This will cause some key macros to be rendered different. The default value is <code>mac</code>.</p>

## 4 Usage

### 4.1 Basics

<code>\menu</code>	<code>\menu{&lt;menu sequence&gt;}</code> , with <code>&gt;</code> as default input list separator, <code>\directory{&lt;path and files&gt;}</code> with <code>/</code> as default separator and <code>\keys{&lt;keystrokes&gt;}</code> with <code>+</code> as default separator. You've seen examples for all of them in section 1.
<code>\directory</code>	
<code>\keys</code>	

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>}`.<sup>3</sup>

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

<sup>2</sup> If you find a conflict send an e-mail.

<sup>3</sup> 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.

### 4.2.1 Predefined styles

Name: `menus`

`File` `Extras` `Preferences` Single

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

Name: `roundedmenus`

`File` `Extras` `Preferences` Single

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

Name: `angularmenus`

`File` `Extras` `Preferences` Single

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

Name: `roundedkeys`

`Ctrl` + `Alt` + `Q` Single

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

Name: `shadowedroundedkeys`

`Ctrl` + `Alt` + `Q` Single

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

Name: **angularkeys**

**Ctrl** + **Alt** + **Q**

**Single**

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

Name: **shadowedangularkeys**

**Ctrl** + **Alt** + **Q**

**Single**

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

Name: **typewriterkeys**

**Alt** + **Q**

**Single**

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

Name: **paths**

**C: ▶ User ▶ Folder ▶ MyFile.tex**

**Single**

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

Name: **pathswithfolder**

**C: ▶ User ▶ Folder ▶ MyFile.tex**

**Single**

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

Name: **pathswithblackfolder**

**C: ▶ User ▶ Folder ▶ MyFile.tex**

**Single**

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

The following three styles allow paths elements to be 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.

Name: `hyphenatepaths`

`C: ▶ Database ▶ User ▶ ALongUserNameHere ▶ ALongerFolderNameAtThisPlace ▶ My  
File.tex` `Single`

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

Name: `hyphenatepathswithfolder`

`☐ C: ▶ Database ▶ User ▶ ALongUserNameHere ▶ ALongerFolderNameAtThisPlace ▶  
MyFile.tex` `☐ Single`

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

Name: `hyphenatepathswithblackfolder`

`▣ C: ▶ Database ▶ User ▶ ALongUserNameHere ▶ ALongerFolderNameAtThisPlace ▶  
MyFile.tex` `▣ Single`

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

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

#### 4.2.2 Declaring styles

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

**name** is the name of the new style. It must follow the specifications of  $\text{\TeX}$  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  $\text{\TikZ}$ -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 it's elements and separates them by a star. We can use

```
\newmenustyle{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 T<sub>E</sub>X 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 TikZ-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 TikZ-style.

**last** is the style for the last list element. It has to be a TikZ-style.

**single** this style is used if the list contains only one element. It has to be a TikZ-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 you needs there are the **starred versions**: Use them and the arguments *<first>*, *<mid>*, *<last>*, *<single>* can be any L<sup>A</sup>T<sub>E</sub>X 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 TikZ **node**. The separator should be the star again:

```
\newmenustyle{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` The next level is redefining a style. This package provides the following macros the work like their L<sup>A</sup>T<sub>E</sub>X-paragons and have the same arguments as the above described macros: `\renewmenustylesimple`, `\providemenustylesimple`, `\renewmenustyle` and `\providemenustyle`.

## 4.3 Color themes

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

### 4.3.1 Predefined themes

There are two predefined color themes

Name: `gray`      Background       Border       Text 

Name: `blackwhite`      Background       Border       Text 

### 4.3.2 Create a theme

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

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

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

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

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

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

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

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

### 4.3.3 Copy a theme

`\copymenucolortheme` To copy the definitions of one theme to another, use `\copymenucolortheme{<copy>}{<original>}`.

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

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

### 4.3.4 Change a theme

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

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

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

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

## 4.4 Menu macros

The "menu 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

`\newmenuacro` To define a new menu macro call `\newmenuacro{<macro>} [<input sep>]{<style>}`.

**name** is a L<sup>A</sup>T<sub>E</sub>X 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

```
\newmenuacro{\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 `\newmenuacro`.

`\providemenuacro` There are also the two commands `\providemenuacro` and `\renewmenuacro` which take the same arguments as `\newmenuacro` and work like their L<sup>A</sup>T<sub>E</sub>X analogons.

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

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

#### 4.5 Keys

`\shift` 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.

`\capslock` macros like `\<key>mac` and `\<key>win` that will always give the right symbol.

`\tab` Here is a full list of available macros:

`\esc`

`\ctrl`

`\alt`

`\AltGr`

`\cmd`

`\Space`

`\return`

`\enter`

`\winmenu`

`\backspace`

`\del`

`\arrowkeyup`

`\arrowkeydown`

`\arrowkeyleft`

`\arrowkeyright`

Macro	Mac	Win.	Macro	Mac	Win.
\shift	⇧	⇧	\winmenu		☰
\capslock	⇞	⇩	\backspace	←	←
\tab	→	⇐→	\del	Del. / ☒	Del.
\esc	esc / ⌘	Esc	\arrowkey{^}	↑	↑
\ctrl	ctrl	Ctrl	\arrowkeyup	↑	↑
\Alt	alt / ⌥	Alt	\arrowkey{v}	↓	↓
\AltGr		Alt Gr	\arrowkeydown	↓	↓
\cmd	cmd / ⌘		\arrowkey{>}	→	→
\Space			\arrowkeyright	→	→
\return	↵	↵	\arrowkey{<}	←	←
\enter	↵	Enter	\arrowkeyleft	←	←

\arrowkey	The macro <code>\arrowkey{&lt;direction&gt;}</code> is a little special since it takes the direction as a single character ^, v (lower case v), > or <.
\ctrlname	The texts for <code>\ctrl</code> and <code>\del</code> are saved in <code>\ctrlname</code> and <code>\delname</code> respectively. So you can change them with <code>\renewcommand</code> .
\delname	
mackeys (opt.)	The rendering of some Mac macros depend on the option <code>mackeys</code> The different versions are shown in the table above (left: <code>text</code> , right: <code>symbols</code> ).

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

## 5 Known issues and bugs

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

**Example** Set `xcolor` to `cmyk` model:

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

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

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

## 6 Implementation

### 6.1 Required packages

Load the required packages

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

Furthermore we need TikZ and some of its libraries,

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

the color package xcolor and adjustbox for the typewriterkeys style.

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

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

```
8 \RequirePackage{relsize}
```

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

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

### 6.2 Helper macros

\tw@mk@error Define macros to call \PackageError and warnings

```
\tw@mk@warning 10 \newcommand*\tw@mk@error[2][Please consult the manual for more information.]{%
\tw@mk@warning@noline 11 \PackageError{menukeys}{#2}{#1}%
12 }
13 \newcommand*\tw@mk@warning[1]{%
14 \PackageWarning{menukeys}{#1}%
15 }
16 \newcommand*\tw@mk@warning@noline[1]{%
17 \PackageWarningNoLine{menukeys}{#1}%
18 }
```

\tw@mk@tempa Some commands for temporary use:

```
\tw@mk@tempb 19 \def\tw@mk@tempa{}
20 \def\tw@mk@tempb{}
```

\tw@mk@gobble@args Define a command to gobble arguments.

```
21 \DeclareDocumentCommand{\tw@mk@gobble@args}{m}{%
22 \RenewDocumentCommand{\tw@mk@tempa}{#1}{}%
23 \tw@mk@tempa%
24 }
```

## 6.3 Options

First we declare and process the package options

```
25 \RequirePackage{kvoptions}
26 \SetupKeyvalOptions{
27   family=tw@mk,
28   prefix=tw@mk@
29 }
30 \DeclareBoolOption[true]{definenummacros}
31 \DeclareBoolOption[true]{definekeys}
32 \DeclareStringOption[mac]{os}
33 \DeclareStringOption[symbols]{mackeys}
34 \ProcessKeyvalOptions{tw@mk}\relax
```

Now we have to do some error treatment:

```
35 \IfSubStr{.mac.win.}{.\tw@mk@os.}{}{\%
36   \tw@mk@error{Unknown value for option 'os'\MessageBreak
37   Possible values are 'mac' or 'win'.}%
38 }
39 \IfSubStr{.symbols.text.}{.\tw@mk@mackeys.}{}{\%
40   \tw@mk@error{Unknown value for option 'mackeys'\MessageBreak
41   Possible values are 'symbols' or 'text'.}%
42 }
```

## 6.4 Color themes

### 6.4.1 Internal commands

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

```
43 \newcommand*{\tw@make@color@theme}[5]{\%
44   \definecolor{tw@color@theme@#1@bg}{#2}{#3}%
45   \definecolor{tw@color@theme@#1@br}{#2}{#4}%
46   \definecolor{tw@color@theme@#1@txt}{#2}{#5}%
47 }
```

### 6.4.2 User-level commands

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

```
\newmenucolortheme 48 \newcommand*{\newmenucolortheme}[5]{\%
49   \@ifundefinedcolor{tw@color@theme@#1@bg}{\%
50     \tw@make@color@theme{#1}{#2}{#3}{#4}{#5}
51   }{\%
52     \tw@mk@error{Color theme '#1' already defined!\MessageBreak
53     Use \string\renewmenucolortheme\space instead.}
54   }
55 }
56 \let\renewmenucolortheme\tw@make@color@theme
```

`\changemenucolor` Lastly we define the changing and copying commands  
`\copymenucolortheme`

```

57 \newcommand*\changemenucolor}[4]{%
58   \IfSubStr{ bg br txt }{ #2 }{%
59     \definecolor{tw@color@theme@#1@#2}{#3}{#4}%
60   }{%
61     \tw@mk@error{No such color element ('#2')!\MessageBreak
62     Possible values are bg, br and txt.}
63   }%
64 }
65 \newcommand*\copymenucolortheme}[2]{%
66   \ifundefinedcolor{tw@color@theme@#1@bg}{%
67     \colorlet{tw@color@theme@#1@bg}{tw@color@theme@#2@bg}%
68     \colorlet{tw@color@theme@#1@br}{tw@color@theme@#2@br}%
69     \colorlet{tw@color@theme@#1@txt}{tw@color@theme@#2@txt}%
70   }{%
71     \tw@mk@error{Color theme '#1' already defined!\MessageBreak
72     Use \string\renewmenucolortheme\space instead.}
73   }
74 }

```

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

```

75 \newcommand*\changemenucolortheme}[2]{%
76   \ifcsundef{tw@style@#1@pre}{%
77     \tw@mk@error{Style '#1' undefined!\MessageBreak
78     Maybe you misspelled it?}%
79   }{%
80     \ifundefinedcolor{tw@color@theme@#2@bg}{%
81       \tw@mk@error{Color theme '#2' is not defined!}%
82     }{%
83       \csdef{tw@style@#1@color@theme}{#2}%
84     }%
85   }%
86 }

```

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

```

87 \newcommand*\usemenucolor}[1]{%
88   tw@color@theme@\tw@current@color@theme @#1%
89 }

```

### 6.4.3 Predefined themes

There are two predefined color themes

```

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

```

## 6.5 Styles

The style generating commands will set some commands that are named like `\tw@style@<name>@<element>`.



<code>\tw@default@sep</code> <code>\tw@default@pre</code> <code>\tw@default@post</code>	<p>Before we can define the internal declaring macro to use it later in the the user level commands, we have to set some defaults for the optional arguments</p> <pre> 92 \newcommand{\tw@default@sep}{% 93   \hspace{0.2em plus 0.1em minus 0.5em}% 94 } 95 \newcommand{\tw@default@pre}{} 96 \newcommand{\tw@default@post}{} </pre>
---	---

### 6.5.1 Internal commands

Now we can define the internal commands.

<code>\tw@declare@style@simple</code>	<p>Our first step is to define the simple command.</p>
---------------------------------------	--

```

97 \DeclareDocumentCommand{\tw@declare@style@simple}{%
98   s m O{\tw@default@pre} m O{\tw@default@sep} O{\tw@default@post} m
99 }{%
100   \csdef{tw@style@#2@color@theme}{#7}%
101   \csdef{tw@style@#2@pre}{#3}%
102   \csdef{tw@style@#2@sep}{#5}%
103   \csdef{tw@style@#2@post}{#6}%
104   \IfBooleanTF{#1}{%
105     \csdef{tw@style@#2@single}{#4}%
106     \csdef{tw@style@#2@first}{#4}%
107     \csdef{tw@style@#2@mid}{#4}%
108     \csdef{tw@style@#2@last}{#4}%
109   }{%
110     \csdef{tw@style@#2@single}{%
111       \tikz[baseline=(tw@node.base)]%
112       \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
113     \csdef{tw@style@#2@first}{%
114       \tikz[baseline=(tw@node.base)]%
115       \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
116     \csdef{tw@style@#2@mid}{%
117       \tikz[baseline=(tw@node.base)]%
118       \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
119     \csdef{tw@style@#2@last}{%
120       \tikz[baseline=(tw@node.base)]%
121       \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
122   }%
123 }

```

<code>\tw@declare@sytle</code> <code>\tw@declare@sytle@extra@args</code>	<p>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.</p>
---	---

```

124 \DeclareDocumentCommand{\tw@declare@sytle@extra@args}{%
125   O{\tw@default@post} m
126 }{%
127   \csdef{tw@style@\tw@current@style @post}{#1}%
128   \csdef{tw@style@\tw@current@style @color@theme}{#2}%

```

129 }

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

```

130 \DeclareDocumentCommand{\tw@declare@style}{%
131   s m O{\tw@default@pre} m O{\tw@default@sep} m m m
132 }{%
133   \def\tw@current@style{#2}
134   \csdef{tw@style@#2@pre}{#3}%
135   \csdef{tw@style@#2@sep}{#5}%
136   \IfBooleanTF{#1}{%
137     \csdef{tw@style@#2@single}{#8}%
138     \csdef{tw@style@#2@first}{#4}%
139     \csdef{tw@style@#2@mid}{#6}%
140     \csdef{tw@style@#2@last}{#7}%
141   }{%
142     \csdef{tw@style@#2@single}{%
143       \tikz[baseline=(tw@node.base)]%
144       \node(tw@node)[#8]{\strut\CurrentMenuElement};}%
145     \csdef{tw@style@#2@first}{%
146       \tikz[baseline=(tw@node.base)]%
147       \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
148     \csdef{tw@style@#2@mid}{%
149       \tikz[baseline=(tw@node.base)]%
150       \node(tw@node)[#6]{\strut\CurrentMenuElement};}%
151     \csdef{tw@style@#2@last}{%
152       \tikz[baseline=(tw@node.base)]%
153       \node(tw@node)[#7]{\strut\CurrentMenuElement};}%
154   }%
155   \tw@declare@style@extra@args%
156 }
```

## 6.5.2 User-level commands

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

```

renewmenustylesimple 157 \NewDocumentCommand{\newmenustylesimple}{s m}{%
providemenustylesimple 158   \ifcsundef{tw@style@#2@pre}{%
newmenustyle 159     \IfBooleanTF{#1}{%
renewmenustyle 160       \tw@declare@style@simple*{#2}%
providemenustyle 161     }{%
162       \tw@declare@style@simple{#2}%
163     }%
164   }{%
165     \tw@mk@error{Style '#2' already defined!\MessageBreak
166     Use \string\renewmenustylesimple\space instead.}%
167     \tw@mk@gobble@args{o m o o m}%
168   }%
169 }
170 \NewDocumentCommand{\renewmenustylesimple}{s m}{%
171   \IfBooleanTF{#1}{%
172     \tw@declare@style@simple*{#2}%

```

```

173 }{%
174 \tw@declare@style@simple{#2}%
175 }%
176 }
177 \NewDocumentCommand{\providemenustylesimple}{s m}{%
178 \ifcsundef{tw@style@#2@pre}{%
179 \IfBooleanTF{#1}{%
180 \tw@declare@style@simple*{#2}%
181 }{%
182 \tw@declare@style@simple{#2}%
183 }%
184 }{%
185 \tw@mk@warning{Trying to provide style ' #2 ' failed,\MessageBreak
186 because it's already defined.\MessageBreak
187 You may use \string\renewmenustylesimple\space instead.}%
188 \tw@mk@gobble@args{o m o o m}%
189 }%
190 }
191
192 \NewDocumentCommand{\newmenustyle}{s m}{%
193 \ifcsundef{tw@style@#2@pre}{%
194 \IfBooleanTF{#1}{%
195 \tw@declare@style*{#2}%
196 }{%
197 \tw@declare@style{#2}%
198 }%
199 }{%
200 \tw@mk@error{Style ' #2 ' already defined!\MessageBreak
201 Use \string\renewmenustyle\space instead.}%
202 \tw@mk@gobble@args{o m o m m m o m}%
203 }%
204 }
205 \NewDocumentCommand{\renewmenustyle}{s m}{%
206 \IfBooleanTF{#1}{%
207 \tw@declare@style*{#2}%
208 }{%
209 \tw@declare@style{#2}%
210 }%
211 }
212 \NewDocumentCommand{\providemenustyle}{s m}{%
213 \ifcsundef{tw@style@#2@pre}{%
214 \IfBooleanTF{#1}{%
215 \tw@declare@style*{#2}%
216 }{%
217 \tw@declare@style{#2}%
218 }%
219 }{%
220 \tw@mk@warning{Trying to provide style #2 failed,\MessageBreak
221 because it's already defined.\MessageBreak
222 You may use \string\renewmenustyle\space instead.}%

```

```

223     \tw@mk@gobble@args{o m o m m m o m}%
224   }%
225 }

```

### 6.5.3 Copying and changing

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

```

226 \newcommand*{\copymenustyle}[2]{%
227   \ifcsundef{tw@style@#1@pre}{%
228     \ifcsundef{tw@style@#2@pre}{%
229       \tw@mk@error{Can't copy not existing style ('#2')}!}%
230     }{%
231       \csletcs{tw@style@#1@pre}{tw@style@#2@pre}%
232       \csletcs{tw@style@#1@post}{tw@style@#2@post}%
233       \csletcs{tw@style@#1@sep}{tw@style@#2@sep}%
234       \csletcs{tw@style@#1@single}{tw@style@#2@single}%
235       \csletcs{tw@style@#1@first}{tw@style@#2@first}%
236       \csletcs{tw@style@#1@mid}{tw@style@#2@mid}%
237       \csletcs{tw@style@#1@last}{tw@style@#2@last}%
238       \csletcs{tw@style@#1@color@theme}{tw@style@#2@color@theme}%
239     }%
240   }{%
241     \tw@mk@error{Style '#1' already exists!}%
242   }%
243 }

```

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

```

244 \NewDocumentCommand{\changemenuelement}{s m m m}{%
245   \ifcsundef{tw@style@#2@pre}{%
246     \tw@mk@error{Style '#2' undefined.}%
247   }{%
248     \IfSubStr{ single first middle last pre post sep }{ #3 }{%
249       \IfBooleanTF{#1}{%
250         \csdef{tw@style@#2@#3}{#4}%
251       }{%
252         \IfSubStr{ pre post sep }{ #3 }{%
253           \csdef{tw@style@#2@#3}{#4}%
254         }{%
255           \csdef{tw@style@#2@#3}{%
256             \tikz[baseline=(tw@node.base)]%
257             \node(tw@node)[#4]{\strut\CurrentMenuElement};}%
258           }%
259         }%
260       }{\tw@mk@error{No element '#3'. Possible values are\MessageBreak
261         single, first, middle, last, pre, post or sep.}}%
262     }%
263 }

```

### 6.5.4 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

```
264 \tikzstyle{tw@set@tikz@colors}=[%
265   draw=\usemenucolor{br},
266   fill=\usemenucolor{bg},
267   text=\usemenucolor{txt},
268 ]
```

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.

```
269 \tikzstyle{tw@menus@base}=[%
270   tw@set@tikz@colors,
271   rounded corners=0.15ex,
272   inner sep=0pt,
273   inner xsep=2pt,
274   text height=1.825ex,
275   text depth=0.7ex,
276   minimum width=1.5em,
277   font=\relsize{-1}\sffamily,
278   signal,
279   signal to=nowhere,
280   signal pointer angle=110,
281 ]
282 \tw@declare@style*{menus}{%
283   \tikz[baseline={{$(tw@node.base)+(0,-0.2ex)$}}]%
284     \node(tw@node)[tw@menus@base,signal to=east]%
285     {\strut\CurrentMenuElement};%
286 }[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]%
287 {%
288   \tikz[baseline={{$(tw@node.base)+(0,-0.2ex)$}}]%
289     \node(tw@node)[tw@menus@base,signal from=west,signal to=east]%
290     {\strut\CurrentMenuElement};%
291 }{%
292   \tikz[baseline={{$(tw@node.base)+(0,-0.2ex)$}}]%
293     \node(tw@node)[tw@menus@base,signal from=west,]%
294     {\strut\CurrentMenuElement};%
295 }{%
296   \tikz[baseline={{$(tw@node.base)+(0,-0.2ex)$}}]%
297     \node(tw@node)[tw@menus@base]{\strut\CurrentMenuElement};%
298 }{gray}
299
300 \tikzstyle{tw@roundedmenus@base}=[%
301   tw@set@tikz@colors,
302   rounded corners=0.3ex,
303   inner sep=0pt,
304   inner xsep=2pt,
305   text height=1.825ex,
```

```

306 text depth=0.7ex,
307 minimum width=1.5em,
308 font=\relsize{-1}\sffamily,
309 signal,
310 signal to=nowhere,
311 signal pointer angle=110,
312 ]
313 \tw@declare@style*{roundedmenus}{%
314   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
315     \node(tw@node)[tw@roundedmenus@base,signal to=east]%
316       {\strut\CurrentMenuElement};%
317 }[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]]%
318 {%
319   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
320     \node(tw@node)[tw@roundedmenus@base,signal from=west,signal to=east]%
321       {\strut\CurrentMenuElement};%
322 }{%
323   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
324     \node(tw@node)[tw@roundedmenus@base,signal from=west,]%
325       {\strut\CurrentMenuElement};%
326 }{%
327   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
328     \node(tw@node)[tw@roundedmenus@base]{\strut\CurrentMenuElement};%
329 }{gray}
330
331 \tikzstyle{tw@angularmenus@base}=[%
332   tw@set@tikz@colors,
333   inner sep=0pt,
334   inner xsep=2pt,
335   text height=1.825ex,
336   text depth=0.7ex,
337   minimum width=1.5em,
338   font=\relsize{-1}\sffamily,
339   signal,
340   signal to=nowhere,
341   signal pointer angle=110,
342 ]
343 \tw@declare@style*{angularmenus}{%
344   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
345     \node(tw@node)[tw@angularmenus@base,signal to=east]%
346       {\strut\CurrentMenuElement};%
347 }[\hspace{-0.2em}\hspace{0em plus 0.1em minus 0.05em}]]%
348 {%
349   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
350     \node(tw@node)[tw@angularmenus@base,signal from=west,signal to=east]%
351       {\strut\CurrentMenuElement};%
352 }{%
353   \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]]%
354     \node(tw@node)[tw@angularmenus@base,signal from=west,]%
355       {\strut\CurrentMenuElement};%

```

```

356 }{%
357   \tikz[baseline={($ (tw@node.base)+(0,-0.2ex)$)}]{}%
358     \node(tw@node)[tw@angularmenus@base]{\strut\CurrentMenuElement};%
359 }{gray}
360
361 \tikzstyle{tw@roundedkeys@base}=[%
362   tw@set@tikz@colors,
363   rounded corners=0.3ex,
364   inner sep=0pt,
365   inner xsep=2pt,
366   text height=1.825ex,
367   text depth=0.7ex,
368   minimum width=1.5em,
369   font=\relsize{-1}\sffamily,
370 ]
371 \tw@declare@style@simple*{roundedkeys}{%
372   \tikz[baseline={($ (tw@node.base)+(0,-0.2ex)$)}]{}%
373     \node(tw@node)[tw@roundedkeys@base]{\strut\CurrentMenuElement};%
374 }[%
375   \hspace{0.1em plus 0.1em minus 0.05em}%
376   \raisebox{0.25ex}{\sffamily\footnotesize}%
377   \hspace{0.1em plus 0.1em minus 0.05em}%
378 ]{gray}
379
380 \tikzstyle{tw@shadowedroundedkeys@base}=[%
381   tw@set@tikz@colors,
382   rounded corners=0.3ex,
383   inner sep=0pt,
384   inner xsep=2pt,
385   text height=1.825ex,
386   text depth=0.7ex,
387   minimum width=1.5em,
388   font=\relsize{-1}\sffamily,
389   general shadow={%
390     shadow xshift=.2ex, shadow yshift=-.15ex,
391     fill=black,
392   },
393 ]
394 \tw@declare@style@simple*{shadowedroundedkeys}{%
395   \tikz[baseline={($ (tw@node.base)+(0,-0.2ex)$)}]{}%
396     \node(tw@node)[tw@shadowedroundedkeys@base]{\strut\CurrentMenuElement};%
397 }[%
398   \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
399   \raisebox{0.25ex}{\sffamily\footnotesize}%
400   \hspace{0.1em plus 0.1em minus 0.05em}%
401 ][\hspace{0.2ex}]{gray}
402
403 \tikzstyle{tw@angularkeys@base}=[%
404   tw@set@tikz@colors,
405   inner sep=0pt,

```

```

406     inner xsep=2pt,
407     text height=1.825ex,
408     text depth=0.7ex,
409     minimum width=1.5em,
410     font=\relsize{-1}\sffamily,
411 ]
412 \tw@declare@style@simple*{angularkeys}{%
413     \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]{%
414         \node(\tw@node)[tw@angularkeys@base]{\strut\CurrentMenuElement};%
415 }[%
416     \hspace{0.1em plus 0.1em minus 0.05em}%
417     \raisebox{0.25ex}{\sffamily\footnotesize}%
418     \hspace{0.1em plus 0.1em minus 0.05em}%
419 ]{gray}
420
421 \tikzstyle{tw@shadowedangularkeys@base}=[%
422     tw@set@tikz@colors,
423     inner sep=0pt,
424     inner xsep=2pt,
425     text height=1.825ex,
426     text depth=0.7ex,
427     minimum width=1.5em,
428     font=\relsize{-1}\sffamily,
429     general shadow={%
430         shadow xshift=.2ex, shadow yshift=-.15ex,
431         fill=black,
432     },
433 ]
434 \tw@declare@style@simple*{shadowedangularkeys}{%
435     \tikz[baseline={($\tw@node.base)+(0,-0.2ex)$}]{%
436         \node(\tw@node)[tw@shadowedangularkeys@base]{\strut\CurrentMenuElement};%
437 }[%
438     \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
439     \raisebox{0.25ex}{\sffamily\footnotesize}%
440     \hspace{0.1em plus 0.1em minus 0.05em}%
441 ][\hspace{0.2ex}]{gray}
442
443 \tikzstyle{tw@typewriterkeys@base}=[%
444     tw@set@tikz@colors,
445     shape=circle,
446     minimum size=2ex,
447     inner sep=0.5pt, outer sep=1pt,
448     font=\ttfamily\relsize{-1},
449 ]
450 \tw@declare@style@simple*{typewriterkeys}{%
451     \def\tw@typewriterkeys@curr@elem{%
452         \maxsizebox*{2ex}{2ex}{\CurrentMenuElement}%
453     }%
454     \begin{tikzpicture}[baseline={($\tw@node.south)+(0,0.8ex)$}]{%
455         \node(\tw@node){%

```



```

456         tw@typewriterkeys@base, inner sep=1.25pt, line width=0.6pt%
457     ]{\tw@typewriterkeys@curr@elem};
458     \node[tw@typewriterkeys@base]{\tw@typewriterkeys@curr@elem};
459     \end{tikzpicture}%
460 }[%
461     \hspace{0.2ex}\hspace{0.1em plus 0.1em minus 0.05em}%
462     \raisebox{0.25ex}{\sffamily\footnotesize+}%
463     \hspace{0.1em plus 0.1em minus 0.05em}%
464 ]{\blacknwhite}
465
466 \tw@declare@style@simple*{paths}{%
467     {\ttfamily\CurrentMenuElement}%
468 }[%
469     \hspace{0.2em plus 0.1em}%
470     \raisebox{0.08ex}{%
471         \tikz\fill[\usemenucolor{txt}] (0,0) -- (0.5ex,0.5ex)%
472         -- (0,1ex) -- cycle;%
473     }%
474     \hspace{0.2em plus 0.1em}%
475 ]{\blacknwhite}
476
477 \newcounter{tw@hyphen@char@num}
478 \newif\if@tw@hyphenatepaths@warnig
479 \@tw@hyphenatepaths@warnigtrue
480 \tw@declare@style@simple*{hyphenatepaths}{%
481     {\ttfamily
482         \IfStrEq{T1}{\encodingdefault}{%
483             \setcounter{tw@hyphen@char@num}{23}%
484         }{%
485             \IfStrEq{OT1}{\encodingdefault}{%
486                 \setcounter{tw@hyphen@char@num}{255}%
487             }{%
488                 \if@tw@hyphenatepaths@warnig%
489                 \tw@mk@warning{The hyphenatepaths styles will probably only\MessageBreak
490                     work with T1 or OT1 encoding.}%
491                 \fi\global\@tw@hyphenatepaths@warnigfalse%
492             }%
493         }%
494         \hyphenchar\font=\value{tw@hyphen@char@num}\relax
495         \CurrentMenuElement}%
496 }[%
497     \hspace{0.2em plus 0.1em}%
498     \raisebox{0.08ex}{%
499         \tikz\fill[\usemenucolor{txt}] (0,0) -- (0.5ex,0.5ex)%
500         -- (0,1ex) -- cycle;%
501     }%
502     \hspace{0.2em plus 0.1em}%
503 ]{\blacknwhite}
504
505 \newcommand*{\drawtikzfolder}[1][white]{%

```

```

506 \begin{tikzpicture}[rounded corners=0.02ex,scale=0.7]
507 \draw (0,0) -- (1em,0) -- (1em,1.5ex) -- (0.5em,1.5ex) -- %
508 (0.4em,1.7ex) -- (0.1em,1.7ex) -- (0,1.5ex) -- cycle;
509 \draw [fill=#1] (0,0) -- (1em,0) -- (0.85em,1.15ex) -- %
510 ++(-1em,0) -- cycle;
511 \end{tikzpicture}%
512 }
513
514 \copymenustyle{pathswithfolder}{paths}
515 \changemenuelement{pathswithfolder}{pre}{%
516 \textcolor{\usemenucolor{txt}}{\drawtikzfolder}%
517 \hspace{0.2em plus 0.1em}%
518 }
519
520 \copymenustyle{pathswithblackfolder}{paths}
521 \changemenuelement{pathswithblackfolder}{pre}{%
522 \textcolor{\usemenucolor{txt}}{\drawtikzfolder[\usemenucolor{txt}]}%
523 \hspace{0.2em plus 0.1em}%
524 }
525
526 \copymenustyle{hyphenatepathswithfolder}{hyphenatepaths}
527 \changemenuelement{hyphenatepathswithfolder}{pre}{%
528 \textcolor{\usemenucolor{txt}}{\drawtikzfolder}%
529 \hspace{0.2em plus 0.1em}%
530 }
531
532 \copymenustyle{hyphenatepathswithblackfolder}{hyphenatepaths}
533 \changemenuelement{hyphenatepathswithblackfolder}{pre}{%
534 \textcolor{\usemenucolor{txt}}{\drawtikzfolder[\usemenucolor{txt}]}%
535 \hspace{0.2em plus 0.1em}%
536 }

```

## 6.6 Menu macros

### 6.6.1 Internal commands

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

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

`\CurrentMenuElement` and the `\CurrentMenuElement` dummy

```
538 \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!

```

539 \begingroup
540 \lccode'\,=1
541 \lowercase{\endgroup
542 \robust@def*\tw@mk@test@input@sep#1{%

```

```

543 \xifinsetTF{\cpttrimspaces{#1},}{,bslash,backslash,directory,location,}%
544 }%
545 }
546 \NewDocumentCommand{\tw@define@menu@macro}{%
547 m O{\tw@default@input@sep} m
548 }{%
549 \ifcsundef{tw@style@#3@sep}{%
550 \tw@mk@error{Can't define menu macro \string#1\space,\MessageBreak
551 because the style '#3' is not available!}%
552 }{%
553 \csdef{tw@parse@menu@list@\expandafter\@gobble\string#1}##1{%
554 \iflastindris
555 \ifnum\indrisnr=\@ne
556 \edef\CurrentMenuElement{##1}%
557 \@nameuse{tw@style@#3@single}%
558 \else
559 \edef\CurrentMenuElement{##1}%
560 \@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@last}%
561 \fi
562 \else
563 \ifnum\indrisnr=\@ne
564 \edef\CurrentMenuElement{##1}%
565 \@nameuse{tw@style@#3@first}%
566 \else
567 \edef\CurrentMenuElement{##1}%
568 \@nameuse{tw@style@#3@sep}\@nameuse{tw@style@#3@mid}%
569 \fi
570 \fi
571 }%
572 \expandafter\newcommand\csname\expandafter\@gobble\string#1\endcsname[2][#2]{%
573 {\def\tw@current@color@theme{\csname tw@style@#3@color@theme\endcsname}%
574 \@nameuse{tw@style@#3@pre}%
575 \tw@mk@test@input@sep{##1}{%
576 \edef\tw@menu@list{\detokenize{##2}}\edef\tw@mk@tempa{\@backslashchar}%
577 }%
578 \edef\tw@menu@list{\unexpanded{##2}}\edef\tw@mk@tempa{\cpttrimspaces{##1}}%
579 }%
580 {\letcs\tw@mk@tempb\tw@parse@menu@list@\expandafter\@gobble\string#1}%
581 \cptexpanded{\indrisloop*[\tw@mk@tempa]\tw@menu@list\tw@mk@tempb}%
582 \@nameuse{tw@style@#3@post}}%
583 }%
584 \expandafter\cproburstify\csname\expandafter\@gobble\string#1\endcsname
585 }%
586 }
587 \edef\cpt@parserlist{\cpt@parserlist\@backslashchar}

```

## 6.6.2 User-level commands

`\newmenumacro` Now it's time to build the user-level commands  
`\renewmenumacro`  
`\providemenumacro`

```

588 \NewDocumentCommand{\newmenumacro}{m O{\tw@default@input@sep} m}{%
589   \ifcsundef{\expandafter\@gobble\string#1}{%
590     \tw@define@menu@macro{#1}{#2}{#3}%
591   }{
592     \tw@mk@error{Menu macro '\string#1' already defined!\MessageBreak
593       Use \string\renewmenustyle\space instead.}
594   }%
595 }
596 \NewDocumentCommand{\renewmenumacro}{m O{\tw@default@input@sep} m}{%
597   \cslet{\expandafter\@gobble\string#1}{\relax}%
598   \tw@define@menu@macro{#1}{#2}{#3}%
599 }
600 \NewDocumentCommand{\providemenumacro}{m O{\tw@default@input@sep} m}{%
601   \ifcsundef{\expandafter\@gobble\string#1}{%
602     \tw@define@menu@macro{#1}{#2}{#3}%
603   }{
604     \tw@mk@warning{Menu macro '\string#1' already defined!\MessageBreak
605       Use \string\renewmenustyle\space to redefine it.}
606   }%
607 }

```

### 6.6.3 Predefined menu macros

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

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

```

\menu And then we define three basic macros.
\directory 609 \newmenumacro{\menu}[>]{menus}
\keys 610 \newmenumacro{\directory}[/]{paths}
611 \newmenumacro{\keys}[+]{roundedkeys}

```

Lastly we close the `definemacros` if statment:

```
612 \fi
```

## 6.7 Keys

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

```
613 \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.

```

614 \NewDocumentCommand{\tw@make@key@box}{m m}{%
615   \expandafter\newbox\csname tw@mk@box@#1\endcsname

```

```

616 \expandafter\sbox\csname tw@mk@box@#1\endcsname{%
617     #2%
618 }%
619 \csdef{tw@mk@#1}{%
620     \expandafter\usebox\csname tw@mk@box@#1\endcsname%
621 }%
622 }

```

`\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.

```

623 \NewDocumentCommand{\tw@make@key@macro}{s m}{%
624     \IfBooleanTF{#1}{%
625         \expandafter\providecommand\csname\expandafter@gobble\string#2\endcsname{%
626             \expandonce{\maxsizebox{!}{1.8ex}}{%
627                 \@nameuse{tw@mk@\expandafter@gobble\string#2@tw@mk@os}}%
628             }%
629         }%
630         \expandafter\providecommand\csname\expandafter@gobble\string#2mac\endcsname{%
631             \expandonce{\maxsizebox{!}{1.8ex}}{%
632                 \@nameuse{tw@mk@\expandafter@gobble\string#2@mac}}%
633             }%
634         }%
635         \expandafter\providecommand\csname\expandafter@gobble\string#2win\endcsname{%
636             \expandonce{\maxsizebox{!}{1.8ex}}{%
637                 \@nameuse{tw@mk@\expandafter@gobble\string#2@win}}%
638             }%
639         }%
640     }{%
641         \expandafter\providecommand\csname\expandafter@gobble\string#2\endcsname{%
642             \expandonce{\maxsizebox{!}{1.8ex}}{%
643                 \@nameuse{tw@mk@\expandafter@gobble\string#2}}%
644             }%
645         }%
646     }%
647 }

```

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

```

648 \newcommand*{\tw@define@mackey}[2]{%
649     \IfStrEq{text}{\tw@mk@mackey}{#1}{%
650         \IfStrEq{symbols}{\tw@mk@mackey}{#2}{%
651             }%
652 }

```

`\shift` Now we are prepared to generate the key macros. I will be nearly the same way  
`\capslock` for all keys. Step one is to build a `tw@mk@⟨key⟩` macro

```

\begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
\ctrl
\alt
\AltGr
\cmd
\Space
\return
\enter
\winmenu
\backspace
\del

```

```

655 \draw (0.3ex,0) -- (1.1ex,0) -- (1.1ex,1.2ex) -- %
656 (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
657 (0.3ex,1.2ex) -- cycle;
658 \end{tikzpicture}%
659 }

```

and then we define the user-level command `\<key>`

```

660 \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`:

```

661 \tw@make@key@box{capslock@mac}{%
662 \begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
663 \draw (0.3ex,0.7ex) -- (1.1ex,0.7ex) -- (1.1ex,1.2ex) -- %
664 (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
665 (0.3ex,1.2ex) -- cycle;
666 \draw (0.3ex,0) rectangle (1.1ex,0.4ex);
667 \end{tikzpicture}%
668 }
669 \tw@make@key@box{capslock@win}{%
670 \begin{tikzpicture}[yscale=-1,yshift=-1.8ex,baseline={(0,0)},semithick]
671 \draw (0.3ex,0) -- (1.1ex,0) -- (1.1ex,1.2ex) -- %
672 (1.5ex,1.2ex) -- (0.7ex,1.9ex) -- (-0.1ex,1.2ex) -- %
673 (0.3ex,1.2ex) -- cycle;
674 \end{tikzpicture}%
675 }

```

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`.

```

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

```

Here are the other macros:

```

677 \tw@make@key@box{tab@mac}{%
678 \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)}]
679 \draw [->,semithick] (0,0) -- (1em,0);
680 \draw (1em,-0.45ex) -- (1em,0.45ex);
681 \end{tikzpicture}%
682 }
683 \tw@make@key@box{tab@win}{%
684 \begin{tikzpicture}[yshift=0.1ex,baseline={(0,0)}]
685 \draw [->,semithick] (0.2em,0) -- (1.2em,0);
686 \draw (1.2em,-0.45ex) -- (1.2em,0.45ex);
687 \draw [<-,semithick] (0,1ex) -- (1em,1ex);
688 \draw (0,0.55ex) -- (0,1.55ex);
689 \end{tikzpicture}%
690 }
691 \tw@make@key@macro*{\tab}
692
693 \def\tw@mk@esc@win{Esc}
694 \tw@define@mackey{%
695 \def\tw@mk@esc@mac{esc}

```

```

696 }{%
697   \tw@make@key@box{esc@mac}{%
698     \begin{tikzpicture}[yshift=-0.1ex,baseline={(0,0)},semithick]
699       \draw [->] (0.5ex,0.5ex) -- ++(45:1.5ex);
700       \draw (0.5ex,0.5ex) ++(15:0.6ex) arc (15:-285:0.6ex);
701     \end{tikzpicture}%
702   }%
703 }
704 \tw@make@key@macro*{\esc}
705
706 \providecommand\ctrlname{Ctrl}
707 \def\tw@mk@ctrl@win{\ctrlname}
708 \def\tw@mk@ctrl@mac{ctrl}
709 \tw@make@key@macro*{\ctrl}
710
711 \def\tw@mk@Alt@win{Alt}
712 \tw@define@mackey{%
713   \def\tw@mk@Alt@mac{alt}%
714 }{%
715   \tw@make@key@box{Alt@mac}{%
716     \begin{tikzpicture}[semithick]
717       \draw (0,1ex) -- (0.5ex,1ex) -- (1ex,0.3ex) -- (1.8ex,0.3ex);
718       \draw (0.8ex,1ex) -- (1.8ex,1ex);
719     \end{tikzpicture}%
720   }%
721 }
722 \tw@make@key@macro*{\Alt}
723
724 \providecommand*{\AltGr}{Alt\,Gr}
725
726 \def\tw@mk@cmd@win{%
727   \tw@mk@warning{'\string\cmd' only for Mac!}%
728 }
729 \tw@define@mackey{%
730   \def\tw@mk@cmd@mac{cmd}%
731 }{%
732   \tw@make@key@box{cmd@mac}{%
733     \begin{tikzpicture}[yshift=-0.15ex,baseline={(0,0)},semithick]
734       \draw (0.5ex,0.7ex) -- (0.5ex,1.25ex) arc (0:270:0.25ex) -- %
735         (1.25ex,1ex) arc (-90:180:0.25ex) -- (1ex,0.25ex) %
736         arc (-180:90:0.25ex) -- (0.25ex,0.5ex) arc (90:360:0.25ex) %
737         -- cycle;
738     \end{tikzpicture}%
739   }%
740 }
741 \tw@make@key@macro*{\cmd}
742
743 \providecommand*{\Space}{\expandonce{\rule{3em}{0pt}}}
744
745 \tw@make@key@box{return@mac}{%

```

```

746 \begin{tikzpicture}[semithick]
747 \draw [->, rounded corners=0.3ex] (1.25ex,1ex) -| %
748 (2ex,0) -- (0,0);
749 \end{tikzpicture}%
750 }
751 \tw@make@key@box{return@win}{%
752 \begin{tikzpicture}[semithick]
753 \draw [->] (1ex,1.25ex) |- (0,0);
754 \end{tikzpicture}%
755 }
756 \tw@make@key@macro*\return}
757
758 \def\tw@mk@enter@win{Enter}
759 \tw@make@key@box{enter@mac}{%
760 \begin{tikzpicture}[semithick]
761 \draw (0,0) -- (0.5ex,0.5ex) -- (1ex,0);
762 \draw (0,0.55ex) -- (1ex,0.55ex);
763 \end{tikzpicture}%
764 }
765 \tw@make@key@macro*\enter}
766
767 \def\tw@mk@winmenu@mac{%
768 \tw@mk@warning{'\string\winmenu' only for Windows!}%
769 }
770 \tw@make@key@box{winmenu@win}{%
771 \begin{tikzpicture}[yshift=-0.2ex,baseline={(0,0)},semithick]
772 \draw (0,0) rectangle (1.5ex,1.8ex);
773 \draw (0.25ex,1.4ex) -- ++(1ex,0);
774 \draw (0.25ex,1ex) -- ++(1ex,0);
775 \draw (0.25ex,0.6ex) -- ++(1ex,0);
776 \end{tikzpicture}%
777 }
778 \tw@make@key@macro*\winmenu}
779
780 \tw@make@key@box{backspace}{%
781 \begin{tikzpicture}[yshift=0.5ex,baseline={(0,0)},thick]
782 \draw [<-] (0,0) -- (1.25em,0);
783 \end{tikzpicture}%
784 }
785 \tw@make@key@macro*\backspace}
786
787 \providecommand{\delname}{Del.}
788 \def\tw@mk@del@win{\delname}
789 \tw@define@mackey{%
790 \def\tw@mk@del@mac{\delname}%
791 }{%
792 \tw@make@key@box{del@mac}{%
793 \begin{tikzpicture}
794 \draw [semithick] (0,0) -- (1.5ex,0) -- (2ex,0.5ex) --%
795 (1.5ex,1ex) -- (0,1ex) -- cycle;

```



```

796         \draw (0.5ex,0.2ex) -- (1.1ex,0.8ex);
797         \draw (0.5ex,0.8ex) -- (1.1ex,0.2ex);
798     \end{tikzpicture}%
799 }%
800 }
801 \tw@make@key@macro*{\del}

\arrowkeyup Lastly we define the arrow macros:
\arrowkeydown 802 \tw@make@key@box{arrowkeyup}{%
\arrowkeyleft 803     \begin{tikzpicture}[yshift=-0.2ex,baseline={{(0,0)}}]
\arrowkeyright 804         \draw [->,semithick] (0,0) -- (0,0.8em);
805     \end{tikzpicture}%
806 }
807 \tw@make@key@macro{\arrowkeyup}
808
809 \tw@make@key@box{arrowkeydown}{%
810     \begin{tikzpicture}[yshift=0.7em,baseline={{(0,0)}}]
811         \draw [->,semithick] (0,0) -- (0,-0.8em);
812     \end{tikzpicture}%
813 }
814 \tw@make@key@macro{\arrowkeydown}
815
816 \tw@make@key@box{arrowkeyright}{%
817     \begin{tikzpicture}[yshift=0.5ex,baseline={{(0,0)}}]
818         \draw [->,semithick] (0,0) -- (0.8em,0);
819     \end{tikzpicture}%
820 }
821 \tw@make@key@macro{\arrowkeyright}
822
823 \tw@make@key@box{arrowkeyleft}{%
824     \begin{tikzpicture}[yshift=0.5ex,baseline={{(0,0)}}]
825         \draw [->,semithick] (0,0) -- (-0.8em,0);
826     \end{tikzpicture}%
827 }
828 \tw@make@key@macro{\arrowkeyleft}

\arrowkey And the \arrowkey macro that get's it's direction as argument.
829 \newcommand{\arrowkey}[1]{%
830     \IfStrEq{^}{#1}{\arrowkeyup}{%
831         \IfStrEq{v}{#1}{\arrowkeydown}{%
832             \IfStrEq{<}{#1}{\arrowkeyleft}{%
833                 \IfStrEq{>}{#1}{\arrowkeyright}{%
834                     \tw@mk@error{Wrong value '#1' for \string\arrowkey\MessageBreak
835                         Possible values are '^', 'v', '<' or '>'}%
836                 }%
837             }%
838         }%
839     }%
840 }

```

Close the \iftw@mk@definekeys  
841 \fi

## 7 Change history

v1.0	Load xcolor before menukeys.	13
General: Initial version	\directory: Renamed \path to	
v1.1	\directory becuase it crashes	
General: Improved manual	with biblatex	28

## 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.

<b>Symbols</b>	
\@tw@hyphenatepaths@warnigfalse	491
\@tw@hyphenatepaths@warnigtrue	479
<b>A</b>	
\Alt	722
\alt	12, 653
\AltGr	12, 653
angularkeys (style)	7
angularmenus (style)	6
\arrowkey	13, 829
\arrowkeydown	12, 802, 831
\arrowkeyleft	12, 802, 832
\arrowkeyright	12, 802, 833
\arrowkeyup	12, 802, 830
<b>B</b>	
\backspace	12, 653
blacknwhite (theme)	10
<b>C</b>	
\capslock	12, 653
\changemenucolor	11, 57
\changemenucolortheme	10, 75
\changemenuelement	10, 244, 515, 521, 527, 533
\cmd	12, 653
Color themes:	
blacknwhite	10
gray	10
\copymenucolortheme	11, 57
\copymenustyle	10, 226, 514, 520, 526, 532
\ctrl	12, 653
\ctrlname	13, 706, 707
\CurrentMenuElement	9, 112, 115, 118, 121, 144, 147, 150, 153, 257, 285, 290, 294, 297, 316, 321, 325, 328, 346, 351, 355, 358, 373, 396, 414, 436, 452, 467, 495, 538, 556, 559, 564, 567
<b>D</b>	
definekeys (option)	5
definenumacros (option)	5
\del	12, 653
\delname	13, 787, 788, 790
\directory	5, 609
\drawtikzfolder	8, 505, 516, 522, 528, 534
<b>E</b>	
\enter	12, 653
\esc	12, 653
<b>F</b>	
\font	494
<b>G</b>	
gray (theme)	10
<b>H</b>	
hyphenatepaths (style)	8
hyphenatepathswithblackfolder (style)	8
hyphenatepathswithfolder (style)	8
<b>I</b>	
\if@tw@hyphenatepaths@warnig	478, 488



<code>\tw@mk@ctrl@mac</code> .....	708	<code>\tw@mk@test@input@sep</code> .....	542, 575
<code>\tw@mk@ctrl@win</code> .....	707	<code>\tw@mk@warning</code> .....	
<code>\tw@mk@del@mac</code> .....	790	..	10, 185, 220, 489, 604, 727, 768
<code>\tw@mk@del@win</code> .....	788	<code>\tw@mk@warning@noline</code> .....	10
<code>\tw@mk@enter@win</code> .....	758	<code>\tw@mk@winmenu@mac</code> .....	767
<code>\tw@mk@error</code> .....	10, 36, 40, 52, 61, 71, 77, 81, 165, 200, 229, 241, 246, 260, 550, 592, 834	<code>\tw@set@tikz@colors</code> .....	264
<code>\tw@mk@esc@mac</code> .....	695	<code>\tw@typewriterkeys@curr@elem</code> ...	451, 457, 458
<code>\tw@mk@esc@win</code> .....	693	<code>typewriterkeys (style)</code> .....	7
<code>\tw@mk@gobble@args</code> .....			
.....	21, 167, 188, 202, 223		
<code>\tw@mk@mackeys</code> .....	39, 649, 650		
<code>\tw@mk@os</code> .....	35, 627		
<code>\tw@mk@tempa</code> .	19, 22, 23, 576, 578, 581		
<code>\tw@mk@tempb</code> .....	19, 580, 581		

## U

<code>\usemenucolor</code> ....	9, 87, 265, 266, 267, 471, 499, 516, 522, 528, 534
---------------------------------	---

## W

<code>\winmenu</code> .....	12, 653
-----------------------------	---------