
KivyMD
Release 1.0.0.dev0

Andrés Rodríguez, Ivanov Yuri, Artem Bulgakov and KivyMD cont

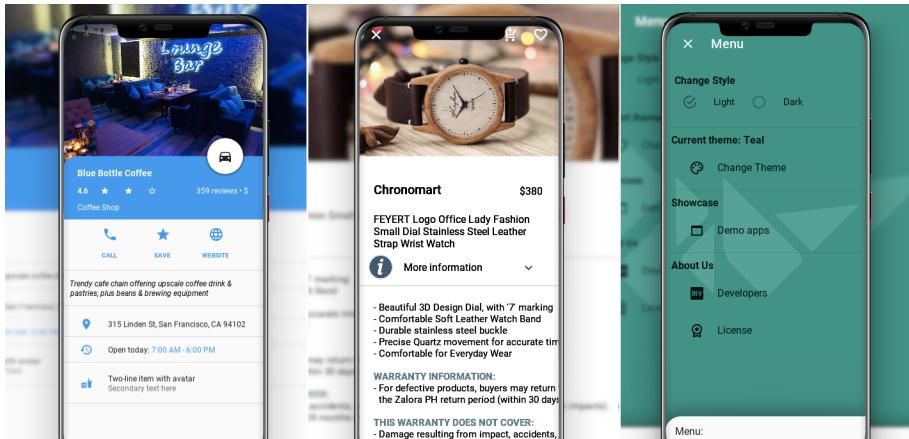
Jul 04, 2021

CONTENTS

1	KivyMD	1
2	Contents	3
2.1	Getting Started	3
2.2	Themes	6
2.3	Components	28
2.4	Behaviors	284
2.5	Changelog	313
2.6	About	321
2.7	KivyMD	322
3	Indices and tables	345
	Python Module Index	347
	Index	349

CHAPTER ONE

KIVYMD



Is a collection of Material Design compliant widgets for use with, Kivy cross-platform graphical framework a framework for cross-platform, touch-enabled graphical applications. The project's goal is to approximate Google's [Material Design spec](#) as close as possible without sacrificing ease of use or application performance.

This library is a fork of the [KivyMD](#) project the author of which stopped supporting this project three years ago. We found the strength and brought this project to a new level. Currently we're in **beta** status, so things are changing all the time and we cannot promise any kind of API stability. However it is safe to vendor now and make use of what's currently available.

Join the project! Just fork the project, branch out and submit a pull request when your patch is ready. If any changes are necessary, we'll guide you through the steps that need to be done via PR comments or access to your for may be requested to outright submit them. If you wish to become a project developer (permission to create branches on the project without forking for easier collaboration), have at least one PR approved and ask for it. If you contribute regularly to the project the role may be offered to you without asking too.

CONTENTS

2.1 Getting Started

In order to start using *KivyMD*, you must first install the *Kivy* framework on your computer. Once you have installed *Kivy*, you can install *KivyMD*.

Warning: *KivyMD* depends on *Kivy*! Therefore, before using *KivyMD*, first learn how to work with *Kivy*.

2.1.1 Installation

```
pip install kivymd
```

Command above will install latest release version of *KivyMD* from PyPI.

If you want to install development version from master branch, you should specify link to zip archive:

```
pip install https://github.com/kivymd/KivyMD/archive/master.zip
```

Tip: Replace *master.zip* with *<commit hash>.zip* (eg *51b8ef0.zip*) to download *KivyMD* from specific commit.

Also you can install manually from sources. Just clone the project and run pip:

```
git clone https://github.com/kivymd/KivyMD.git --depth 1
cd KivyMD
pip install .
```

Speed Tip: If you don't need full commit history (about 320 MiB), you can use a shallow clone (*git clone https://github.com/kivymd/KivyMD.git --depth 1*) to save time. If you need full commit history, then remove *--depth 1*.

2.1.2 First KivyMD application

```
from kivymd.app import MDApp
from kivymd.uix.label import MDLabel

class MainApp(MDApp):
    def build(self):
        return MDLabel(text="Hello, World", halign="center")
```

(continues on next page)

(continued from previous page)

```
MainApp().run()
```

And the equivalent with *Kivy*:

```
from kivy.app import App
from kivy.uix.label import Label

class MainApp(App):
    def build(self):
        return Label(text="Hello, World")

MainApp().run()
```

To left - *Kivy*, to right - *KivyMD*:



At first glance, the *KivyMD* example contains more code... However, the following example already demonstrates how difficult it is to create a custom button in *Kivy*:

```
from kivy.app import App
from kivy.metrics import dp
from kivy.uix.behaviors import TouchRippleBehavior
from kivy.uix.button import Button
from kivy.lang import Builder

KV = """
<RectangleFlatButton>:
    ripple_color: 0, 0, 0, .2
    background_color: 0, 0, 0, 0
```

(continues on next page)

(continued from previous page)

```

color: root.primary_color

canvas.before:
    Color:
        rgba: root.primary_color
    Line:
        width: 1
        rectangle: (self.x, self.y, self.width, self.height)

Screen:
    canvas:
        Color:
            rgba: 0.9764705882352941, 0.9764705882352941, 0.9764705882352941, 1
        Rectangle:
            pos: self.pos
            size: self.size
    .....

class RectangleFlatButton(TouchRippleBehavior, Button):
    primary_color = [
        0.12941176470588237,
        0.5882352941176471,
        0.9529411764705882,
        1
    ]

    def on_touch_down(self, touch):
        collide_point = self.collide_point(touch.x, touch.y)
        if collide_point:
            touch.grab(self)
            self.ripple_show(touch)
            return True
        return False

    def on_touch_up(self, touch):
        if touch.grab_current is self:
            touch.ungrab(self)
            self.ripple_fade()
            return True
        return False

class MainApp(App):
    def build(self):
        screen = Builder.load_string(KV)
        screen.add_widget(
            RectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
                size_hint=(None, None),
                size=(dp(110), dp(35)),

```

(continues on next page)

(continued from previous page)

```
        ripple_color=(0.8, 0.8, 0.8, 0.5),  
    )  
)  
return screen  
  
MainApp().run()
```

And the equivalent with *KivyMD*:

```
from kivy.uix.screenmanager import Screen  
  
from kivymd.app import MDApp  
from kivymd.uix.button import MDRectangleFlatButton  
  
class MainApp(MDApp):  
    def build(self):  
        screen = Screen()  
        screen.add_widget(  
            MDRectangleFlatButton(  
                text="Hello, World",  
                pos_hint={"center_x": 0.5, "center_y": 0.5},  
            )  
        )  
    return screen  
  
MainApp().run()
```

To left - *Kivy*, to right - *KivyMD*:

2.2 Themes

2.2.1 Theming

See also:

Material Design spec, Material theming

Material App

The main class of your application, which in *Kivy* inherits from the `App` class, in *KivyMD* must inherit from the `MDApp` class. The `MDApp` class has properties that allow you to control application properties such as `color/style/font` of interface elements and much more.

Control material properties

The main application class inherited from the `MDApp` class has the `theme_cls` attribute, with which you control the material properties of your application.

Changing the theme colors

The standard `theme_cls` is designed to provide the standard themes and colors as defined by Material Design.

We do not recommend that you change this.

However, if you do need to change the standard colors, for instance to meet branding guidelines, you can do this by overloading the `color_definitions.py` object.

- Create a custom color defintion object. This should have the same format as the `colors` object in `color_definitions.py` and contain definitions for at least the primary color, the accent color and the Light and Dark backgrounds.

Note: Your custom colors *must* use the names of the existing colors as defined in the palette e.g. You can have *Blue* but you cannot have *NavyBlue*.

- Add the custom theme to the `MDApp` as shown in the following snippet.

```
from kivy.lang import Builder
from kivy.properties import ObjectProperty

from kivymd.app import MDApp
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.tab import MDTabsBase
from kivymd.icon_definitions import md_icons

colors = {
    "Teal": {
        "50": "e4f8f9",
        "100": "bdedf0",
        "200": "97e2e8",
        "300": "79d5de",
        "400": "6dcdb6",
        "500": "6ac2cf",
        "600": "63b2bc",
        "700": "5b9ca3",
        "800": "54888c",
        "900": "486363",
        "A100": "bdedf0",
        "A200": "97e2e8",
    }
}
```

(continues on next page)

(continued from previous page)

```

    "A400": "6dcbd6",
    "A700": "5b9ca3",
},
"Blue": {
    "50": "e3f3f8",
    "100": "b9e1ee",
    "200": "91cee3",
    "300": "72bad6",
    "400": "62acce",
    "500": "589fc6",
    "600": "5191b8",
    "700": "487fa5",
    "800": "426f91",
    "900": "35506d",
    "A100": "b9e1ee",
    "A200": "91cee3",
    "A400": "62acce",
    "A700": "487fa5",
},
"Light": {
    "StatusBar": "E0E0E0",
    "AppBar": "F5F5F5",
    "Background": "FAFAFA",
    "CardsDialogs": "FFFFFF",
    "FlatButtonDown": "cccccc",
},
"Dark": {
    "StatusBar": "000000",
    "AppBar": "212121",
    "Background": "303030",
    "CardsDialogs": "424242",
    "FlatButtonDown": "999999",
}
}

KV = '''
BoxLayout:
    orientation: "vertical"
    MDToolbar:
        title: "Example Tabs"
    MDTabs:
        id: tabs

<Tab>

    MDIconButton:
        id: icon
        icon: root.icon
        user_font_size: "48sp"
        pos_hint: {"center_x": .5, "center_y": .5}

```

(continues on next page)

(continued from previous page)

```
"""
    ...

class Tab(MDFloatLayout, MDTabsBase):
    "Class implementing content for a tab."

    icon = ObjectProperty()

class Example(MDApp):
    icons = list(md_icons.keys())[15:30]

    def build(self):
        self.theme_cls.colors = colors
        self.theme_cls.primary_palette = "Blue"
        self.theme_cls.accent_palette = "Teal"
        return Builder.load_string(KV)

    def on_start(self):
        for name_tab in self.icons:
            tab = Tab(text="This is " + name_tab, icon=name_tab)
            self.root.ids.tabs.add_widget(tab)

Example().run()
```

This will change the theme colors to your custom definition. In all other respects, the theming stays as documented.

API - kivymd.theming

`class kivymd.theming.ThemeManager(**kwargs)`

`primary_palette`

The name of the color scheme that the application will use. All major *material* components will have the color of the specified color theme.

Available options are: 'Red', 'Pink', 'Purple', 'DeepPurple', 'Indigo', 'Blue', 'LightBlue', 'Cyan', 'Teal', 'Green', 'LightGreen', 'Lime', 'Yellow', 'Amber', 'Orange', 'DeepOrange', 'Brown', 'Gray', 'BlueGray'.

To change the color scheme of an application:

```
from kivy.uix.screenmanager import Screen

from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Green" # "Purple", "Red"

        screen = Screen()
```

(continues on next page)

(continued from previous page)

```

        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
    return screen

MainApp().run()

```



`primary_palette` is an `OptionProperty` and defaults to ‘Blue’.

primary_hue

The color hue of the application.

Available options are: ‘50’, ‘100’, ‘200’, ‘300’, ‘400’, ‘500’, ‘600’, ‘700’, ‘800’, ‘900’, ‘A100’, ‘A200’, ‘A400’, ‘A700’.

To change the hue color scheme of an application:

```

from kivy.uix.screenmanager import Screen

from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Green" # "Purple", "Red"
        self.theme_cls.primary_hue = "200" # "500"

        screen = Screen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
    return screen

```

(continues on next page)

(continued from previous page)

```
MainApp().run()
```

With a value of `self.theme_cls.primary_hue = "500"`:



With a value of `self.theme_cls.primary_hue = "200"`:



`primary_hue` is an `OptionProperty` and defaults to '500'.

`primary_light_hue`

Hue value for `primary_light`.

`primary_light_hue` is an `OptionProperty` and defaults to '200'.

`primary_dark_hue`

Hue value for `primary_dark`.

`primary_light_hue` is an `OptionProperty` and defaults to '700'.

`primary_color`

The color of the current application theme in `rgba` format.

`primary_color` is an `AliasProperty` that returns the value of the current application theme, property is readonly.

`primary_light`

Colors of the current application color theme in `rgba` format (in lighter color).

```
from kivy.lang import Builder
from kivymd.app import MDApp
```

(continues on next page)

(continued from previous page)

```
KV = '''
Screen:

    MDRaisedButton:
        text: "primary_light"
        pos_hint: {"center_x": 0.5, "center_y": 0.7}
        md_bg_color: app.theme_cls.primary_light

    MDRaisedButton:
        text: "primary_color"
        pos_hint: {"center_x": 0.5, "center_y": 0.5}

    MDRaisedButton:
        text: "primary_dark"
        pos_hint: {"center_x": 0.5, "center_y": 0.3}
        md_bg_color: app.theme_cls.primary_dark
    ...

class MainApp(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Green"
        return Builder.load_string(KV)

MainApp().run()
```



`primary_light` is an [AliasProperty](#) that returns the value of the current application theme (in lighter color), property is readonly.

`primary_dark`

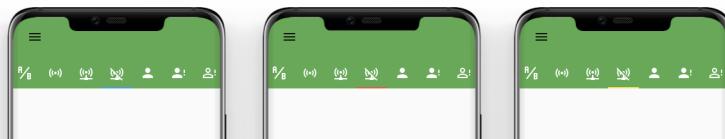
Colors of the current application color theme in `rgba` format (in darker color).

`primary_dark` is an [AliasProperty](#) that returns the value of the current application theme (in darker color), property is readonly.

`accent_palette`

The application color palette used for items such as the tab indicator in the `MDTabsBar` class and so on...

The image below shows the color schemes with the values `self.theme_cls.accent_palette = 'Blue', Red' and Yellow'`:



`accent_palette` is an `OptionProperty` and defaults to ‘Amber’.

`accent_hue`

Similar to `primary_hue`, but returns a value for `accent_palette`.

`accent_hue` is an `OptionProperty` and defaults to ‘500’.

`accent_light_hue`

Hue value for `accent_light`.

`accent_light_hue` is an `OptionProperty` and defaults to ‘200’.

`accent_dark_hue`

Hue value for `accent_dark`.

`accent_dark_hue` is an `OptionProperty` and defaults to ‘700’.

`accent_color`

Similar to `primary_color`, but returns a value for `accent_color`.

`accent_color` is an `AliasProperty` that returns the value in `rgba` format for `accent_color`, property is readonly.

`accent_light`

Similar to `primary_light`, but returns a value for `accent_light`.

`accent_light` is an `AliasProperty` that returns the value in `rgba` format for `accent_light`, property is readonly.

`accent_dark`

Similar to `primary_dark`, but returns a value for `accent_dark`.

`accent_dark` is an `AliasProperty` that returns the value in `rgba` format for `accent_dark`, property is readonly.

`theme_style`

App theme style.

```
from kivy.uix.screenmanager import Screen

from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark" # "Light"

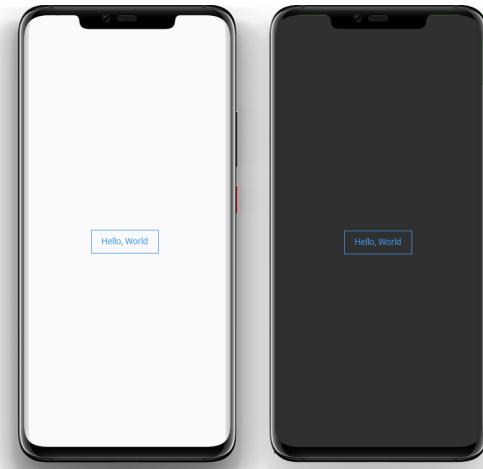
        screen = Screen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
```

(continues on next page)

(continued from previous page)

```
return screen
```

```
MainApp().run()
```



theme_style is an `OptionProperty` and defaults to ‘Light’.

bg_darkest

Similar to `bg_dark`, but the color values are a tone lower (darker) than `bg_dark`.

```
KV = '''
<Box@BoxLayout>:
    bg: 0, 0, 0, 0

    canvas:
        Color:
            rgba: root.bg
        Rectangle:
            pos: self.pos
            size: self.size

BoxLayout:
    Box:
        bg: app.theme_cls.bg_light
    Box:
        bg: app.theme_cls.bg_normal
    Box:
        bg: app.theme_cls.bg_dark
    Box:
        bg: app.theme_cls.bg_darkest
    ...

from kivy.lang import Builder
from kivymd.app import MDApp
```

(continues on next page)

(continued from previous page)

```
class MainApp(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark" # "Light"
        return Builder.load_string(KV)

MainApp().run()
```



bg_darkest is an [AliasProperty](#) that returns the value in `rgba` format for *bg_darkest*, property is readonly.

opposite_bg_darkest

The opposite value of color in the *bg_darkest*.

opposite_bg_darkest is an [AliasProperty](#) that returns the value in `rgba` format for *opposite_bg_darkest*, property is readonly.

bg_dark

Similar to *bg_normal*, but the color values are one tone lower (darker) than *bg_normal*.

bg_dark is an [AliasProperty](#) that returns the value in `rgba` format for *bg_dark*, property is readonly.

opposite_bg_dark

The opposite value of color in the *bg_dark*.

opposite_bg_dark is an [AliasProperty](#) that returns the value in `rgba` format for *opposite_bg_dark*, property is readonly.

bg_normal

Similar to *bg_light*, but the color values are one tone lower (darker) than *bg_light*.

bg_normal is an [AliasProperty](#) that returns the value in `rgba` format for *bg_normal*, property is readonly.

opposite_bg_normal

The opposite value of color in the *bg_normal*.

opposite_bg_normal is an [AliasProperty](#) that returns the value in `rgba` format for *opposite_bg_normal*, property is readonly.

bg_light

” Depending on the style of the theme (‘Dark’ or ‘Light’) that the application uses, `bg_light` contains the color value in `rgba` format for the widgets background.

`bg_light` is an `AliasProperty` that returns the value in `rgba` format for `bg_light`, property is readonly.

opposite_bg_light

The opposite value of color in the `bg_light`.

`opposite_bg_light` is an `AliasProperty` that returns the value in `rgba` format for `opposite_bg_light`, property is readonly.

divider_color

Color for dividing lines such as `MDSeparator`.

`divider_color` is an `AliasProperty` that returns the value in `rgba` format for `divider_color`, property is readonly.

opposite_divider_color

The opposite value of color in the `divider_color`.

`opposite_divider_color` is an `AliasProperty` that returns the value in `rgba` format for `opposite_divider_color`, property is readonly.

text_color

Color of the text used in the `MDLabel`.

`text_color` is an `AliasProperty` that returns the value in `rgba` format for `text_color`, property is readonly.

opposite_text_color

The opposite value of color in the `text_color`.

`opposite_text_color` is an `AliasProperty` that returns the value in `rgba` format for `opposite_text_color`, property is readonly.

secondary_text_color

The color for the secondary text that is used in classes from the module `TwoLineListItem`.

`secondary_text_color` is an `AliasProperty` that returns the value in `rgba` format for `secondary_text_color`, property is readonly.

opposite_secondary_text_color

The opposite value of color in the `secondary_text_color`.

`opposite_secondary_text_color` is an `AliasProperty` that returns the value in `rgba` format for `opposite_secondary_text_color`, property is readonly.

icon_color

Color of the icon used in the `MDIconButton`.

`icon_color` is an `AliasProperty` that returns the value in `rgba` format for `icon_color`, property is readonly.

opposite_icon_color

The opposite value of color in the `icon_color`.

`opposite_icon_color` is an `AliasProperty` that returns the value in `rgba` format for `opposite_icon_color`, property is readonly.

disabled_hint_text_color

Color of the disabled text used in the `MDTextField`.

disabled_hint_text_color is an `AliasProperty` that returns the value in `rgba` format for *disabled_hint_text_color*, property is readonly.

opposite_disabled_hint_text_color

The opposite value of color in the *disabled_hint_text_color*.

opposite_disabled_hint_text_color is an `AliasProperty` that returns the value in `rgba` format for *opposite_disabled_hint_text_color*, property is readonly.

error_color

Color of the error text used in the `MTextField`.

error_color is an `AliasProperty` that returns the value in `rgba` format for *error_color*, property is readonly.

ripple_color

Color of ripple effects.

ripple_color is an `AliasProperty` that returns the value in `rgba` format for *ripple_color*, property is readonly.

device_orientation

Device orientation.

device_orientation is an `StringProperty`.

standard_increment

Value of standard increment.

standard_increment is an `AliasProperty` that returns the value in `rgba` format for *standard_increment*, property is readonly.

horizontal_margins

Value of horizontal margins.

horizontal_margins is an `AliasProperty` that returns the value in `rgba` format for *horizontal_margins*, property is readonly.

set_clearcolor

font_styles

Data of default font styles.

Add custom font:

```
KV = '''
Screen:

    MDLabel:
        text: "JetBrainsMono"
        halign: "center"
        font_style: "JetBrainsMono"
    ...

from kivy.core.text import LabelBase

from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.font_definitions import theme_font_styles
```

(continues on next page)

(continued from previous page)

```

class MainApp(MDApp):
    def build(self):
        LabelBase.register(
            name="JetBrainsMono",
            fn_regular="JetBrainsMono-Regular.ttf")

        theme_font_styles.append('JetBrainsMono')
        self.theme_cls.font_styles["JetBrainsMono"] = [
            "JetBrainsMono",
            16,
            False,
            0.15,
        ]
    return Builder.load_string(KV)

MainApp().run()

```



`font_styles` is an `DictProperty`.

`on_theme_style(self, instance, value)`

`set_clearcolor_by_theme_style(self, theme_style)`

`set_colors(self, primary_palette, primary_hue, primary_light_hue, primary_dark_hue, accent_palette, accent_hue, accent_light_hue, accent_dark_hue)`

Courtesy method to allow all of the theme color attributes to be set in one call.

`set_colors` allows all of the following to be set in one method call:

- primary palette color,
- primary hue,
- primary light hue,
- primary dark hue,
- accent palette color,

- accent hue,
- accent light hue, and
- accent dark hue.

Note that all values *must* be provided. If you only want to set one or two values use the appropriate method call for that.

```
from kivy.uix.screenmanager import Screen

from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.set_colors(
            "Blue", "600", "50", "800", "Teal", "600", "100", "800"
        )

        screen = Screen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
    return screen

MainApp().run()
```

```
sync_theme_styles(self, *args)
class kivymd.theming.ThemableBehavior(**kwargs)

theme_cls
    Instance of ThemeManager class.
    theme_cls is an ObjectProperty.

device_ios
    True if device is iOS.
    device_ios is an BooleanProperty.

widget_style
    Allows to set one of the three style properties for the widget: ‘android’, ‘ios’, ‘desktop’.
```

For example, for the class `MDSwitch` has two styles - ‘`android`’ and ‘`ios`’:

```
MDSwitch:
    widget_style: "ios"
```

```
MDSwitch:  
    widget_style: "android"
```

`widget_style` is an `OptionProperty` and defaults to ‘*android*’.

`opposite_colors`

2.2.2 Material App

This module contains `MDApp` class that is inherited from `App`. `MDApp` has some properties needed for KivyMD library (like `theme_cls`).

You can turn on the monitor displaying the current FPS value in your application:

```
KV = ''''  
Screen:  
  
    MDLabel:  
        text: "Hello, World!"  
        halign: "center"  
    ...  
  
from kivy.lang import Builder  
  
from kivymd.app import MDApp  
  
  
class MainApp(MDApp):  
    def build(self):  
        return Builder.load_string(KV)  
  
    def on_start(self):  
        self.fps_monitor_start()  
  
MainApp().run()
```



API - kivymd.app

```
class kivymd.app.MDApp(**kwargs)
```

Application class, see module documentation for more information.

Events

on_start: Fired when the application is being started (before the `runTouchApp()` call).

on_stop: Fired when the application stops.

on_pause: Fired when the application is paused by the OS.

on_resume: Fired when the application is resumed from pause by the OS. Beware: you have no guarantee that this event will be fired after the `on_pause` event has been called.

Changed in version 1.7.0: Parameter `kv_file` added.

Changed in version 1.8.0: Parameters `kv_file` and `kv_directory` are now properties of App.

theme_cls

Instance of ThemeManager class.

Warning: The `theme_cls` attribute is already available in a class that is inherited from the `MDApp` class. The following code will result in an error!

```
class MainApp(MDApp):
    theme_cls = ThemeManager()
    theme_cls.primary_palette = "Teal"
```

Note: Correctly do as shown below!

```
class MainApp(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Teal"
```

`theme_cls` is an `ObjectProperty`.

2.2.3 Color Definitions

See also:

Material Design spec, The color system

Material Design spec, The color tool

Material colors palette to use in `kivymd.theming.ThemeManager.colors` is a dict-in-dict where the first key is a value from `palette` and the second key is a value from `hue`. Color is a hex value, a string of 6 characters (0-9, A-F) written in uppercase.

For example, `colors["Red"]["900"]` is "B71C1C".

API - kivymd.color_definitions

kivymd.color_definitions.colors

Color palette. Taken from 2014 Material Design color palettes.

To demonstrate the shades of the palette, you can run the following code:

```
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
from kivy.utils import get_color_from_hex
from kivy.properties import ListProperty, StringProperty

from kivymd.color_definitions import colors
from kivymd.uix.tab import MDTabsBase

demo = '''
<Root@MDBoxLayout>
```

(continues on next page)

(continued from previous page)

```

orientation: 'vertical'

MDToolbar:
    title: app.title

MDTabs:
    id: android_tabs
    on_tab_switch: app.on_tab_switch(*args)
    size_hint_y: None
    height: "48dp"
    tab_indicator_anim: False

RecycleView:
    id: rv
    key_viewclass: "viewclass"
    key_size: "height"

RecycleBoxLayout:
    default_size: None, dp(48)
    default_size_hint: 1, None
    size_hint_y: None
    height: self.minimum_height
    orientation: "vertical"

<ItemColor>
    size_hint_y: None
    height: "42dp"

MDLabel:
    text: root.text
    halign: "center"

<Tab>
```
from kivy.factory import Factory
from kivymd.app import MDApp

class Tab(BoxLayout, MDTabsBase):
 pass

class ItemColor(MDBoxLayout):
 text = StringProperty()
 color = ListProperty()

class Palette(MDApp):
 title = "Colors definitions"

```

(continues on next page)

(continued from previous page)

```

def build(self):
 Builder.load_string(demo)
 self.screen = Factory.Root()

 for name_tab in colors.keys():
 tab = Tab(text=name_tab)
 self.screen.ids.android_tabs.add_widget(tab)

 return self.screen

def on_tab_switch(self, instance_tabs, instance_tab, instance_tabs_label, tab_text):
 self.screen.ids.rv.data = []
 if not tab_text:
 tab_text = 'Red'
 for value_color in colors[tab_text]:
 self.screen.ids.rv.data.append(
 {
 "viewclass": "ItemColor",
 "md_bg_color": get_color_from_hex(colors[tab_text][value_color]),
 "text": value_color,
 }
)

def on_start(self):
 self.on_tab_switch(
 None,
 None,
 None,
 self.screen.ids.android_tabs.ids.layout.children[-1].text,
)

Palette().run()

```

kivymd.color\_definitions.palette = ['Red', 'Pink', 'Purple', 'DeepPurple', 'Indigo', 'Blue', 'LightBlue', 'Cyan', 'Teal', 'Green', 'LightGreen', 'Lime', 'Yellow', 'Amber', 'Orange', 'DeepOrange', 'Brown', 'Gray', 'BlueGray']

Valid values for color palette selecting.

kivymd.color\_definitions.hue = ['50', '100', '200', '300', '400', '500', '600', '700', '800', '900', 'A100', 'A200', 'A400', 'A700']

Valid values for color hue selecting.

kivymd.color\_definitions.light\_colors

Which colors are light. Other are dark.

kivymd.color\_definitions.text\_colors

Text colors generated from `light_colors`. “000000” for light and “FFFFFF” for dark.

How to generate text\_colors dict

```

text_colors = {}
for p in palette:
 text_colors[p] = {}
 for h in hue:
 if h in light_colors[p]:
 text_colors[p][h] = "000000"
 else:
 text_colors[p][h] = "FFFFFF"

```

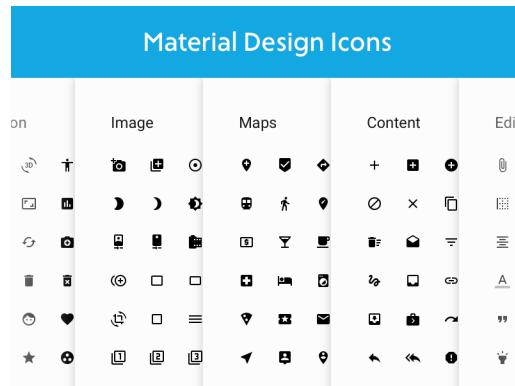
kivymd.color\_definitions.theme\_colors = ['Primary', 'Secondary', 'Background', 'Surface', 'Error', 'On\_Primary', 'On\_Secondary', 'On\_Background', 'On\_Surface', 'On\_Error']

Valid theme colors.

## 2.2.4 Icon Definitions

See also:

[Material Design Icons](#)



List of icons from [materialdesignicons.com](http://materialdesignicons.com). These expanded material design icons are maintained by Austin Andrews (Templarian on Github).

LAST UPDATED: Version 5.9.55

To preview the icons and their names, you can use the following application:

```

from kivy.lang import Builder
from kivy.properties import StringProperty
from kivy.uix.screenmanager import Screen

from kivymd.icon_definitions import md_icons
from kivymd.app import MDApp
from kivymd.uix.list import OneLineIconListItem

Builder.load_string(
"""
#:import images_path kivymd.images_path

```

(continues on next page)

(continued from previous page)

```
<CustomOneLineIconListItem>:

 IconLeftWidget:
 icon: root.icon

<PreviousMDIcons>:

 MDBBoxLayout:
 orientation: 'vertical'
 spacing: dp(10)
 padding: dp(20)

 MDBBoxLayout:
 adaptive_height: True

 MDIconButton:
 icon: 'magnify'

 MDTextField:
 id: search_field
 hint_text: 'Search icon'
 on_text: root.set_list_md_icons(self.text, True)

 RecycleView:
 id: rv
 key_viewclass: 'viewclass'
 key_size: 'height'

 RecycleBoxLayout:
 padding: dp(10)
 default_size: None, dp(48)
 default_size_hint: 1, None
 size_hint_y: None
 height: self.minimum_height
 orientation: 'vertical'
"""

)

class CustomOneLineIconListItem(OneLineIconListItem):
 icon = StringProperty()

class PreviousMDIcons(Screen):

 def set_list_md_icons(self, text="", search=False):
 """Builds a list of icons for the screen MDIcons."""

 def add_icon_item(name_icon):
```

(continues on next page)

(continued from previous page)

```

self.ids.rv.data.append(
 {
 "viewclass": "CustomOneLineIconListItem",
 "icon": name_icon,
 "text": name_icon,
 "callback": lambda x: x,
 }
)

self.ids.rv.data = []
for name_icon in md_icons.keys():
 if search:
 if text in name_icon:
 add_icon_item(name_icon)
 else:
 add_icon_item(name_icon)

class MainApp(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = PreviousMDIcons()

 def build(self):
 return self.screen

 def on_start(self):
 self.screen.set_list_md_icons()

MainApp().run()

```

**API - kivymd.icon\_definitions**

kivymd.icon\_definitions.md\_icons

**2.2.5 Font Definitions****See also:**

Material Design spec, The type system

## API - kivymd.font\_definitions

```
kivymd.font_definitions.fonts
```

```
kivymd.font_definitions.theme_font_styles = ['H1', 'H2', 'H3', 'H4', 'H5', 'H6',
'Subtitle1', 'Subtitle2', 'Body1', 'Body2', 'Button', 'Caption', 'Overline', 'Icon']
```

| Scale Category | Typeface | Font    | Size | Case     | Letter spacing |
|----------------|----------|---------|------|----------|----------------|
| H1             | Roboto   | Light   | 96   | Sentence | -1.5           |
| H2             | Roboto   | Light   | 60   | Sentence | -0.5           |
| H3             | Roboto   | Regular | 48   | Sentence | 0              |
| H4             | Roboto   | Regular | 34   | Sentence | 0.25           |
| H5             | Roboto   | Regular | 24   | Sentence | 0              |
| H6             | Roboto   | Medium  | 20   | Sentence | 0.15           |
| Subtitle 1     | Roboto   | Regular | 16   | Sentence | 0.15           |
| Subtitle 2     | Roboto   | Medium  | 14   | Sentence | 0.1            |
| Body 1         | Roboto   | Regular | 16   | Sentence | 0.5            |
| Body 2         | Roboto   | Regular | 14   | Sentence | 0.25           |
| BUTTON         | Roboto   | Medium  | 14   | All caps | 1.25           |
| Caption        | Roboto   | Regular | 12   | Sentence | 0.4            |
| OVERLINE       | Roboto   | Regular | 10   | All caps | 1.5            |

## 2.3 Components

### 2.3.1 Screen

Screen class equivalent. Simplifies working with some widget properties. For example:

## Screen

```
Screen:
 canvas:
 Color:
 rgba: app.theme_cls.primary_color
 RoundedRectangle:
 pos: self.pos
 size: self.size
 radius: [25, 0, 0]
```

## MDScreen

```
MDScreen:
 radius: [25, 0, 0]
 md_bg_color: app.theme_cls.primary_color
```

### API - kivymd.uix.screen

**class kivymd.uix.screen.MDScreen(\*\*kw)**

Screen is an element intended to be used with a `ScreenManager`. Check module documentation for more information.

#### Events

***on\_pre\_enter*:** () Event fired when the screen is about to be used: the entering animation is started.

***on\_enter*:** () Event fired when the screen is displayed: the entering animation is complete.

***on\_pre\_leave*:** () Event fired when the screen is about to be removed: the leaving animation is started.

***on\_leave*:** () Event fired when the screen is removed: the leaving animation is finished.

Changed in version 1.6.0: Events `on_pre_enter`, `on_enter`, `on_pre_leave` and `on_leave` were added.

## 2.3.2 Menu

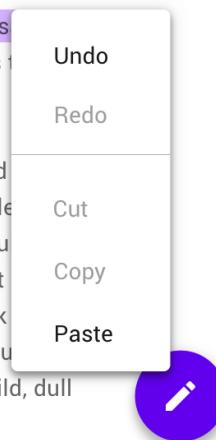
### See also:

Material Design spec, Menus

### Menus display a list of choices on temporary surfaces.

es lay spread out on the table - Samsa was a travelling salesman - and above a picture that he had recently cut out of an illustrated magazine and housed in a frame. It showed a lady fitted out with a fur hat and fur boa who was holding a heavy fur muff that covered the whole of her lower arm towards the end of the story.

urned to look out the window at the dull weather. Drops of rain could be seen falling on the pane, which made him feel quite sad. "How about if I sleep a little longer all this nonsense", he thought, but that was something he was used to sleeping on his right, and in his present state couldn't manage it. However hard he threw himself onto his right, he always rolled back onto his left. He must have tried it a hundred times, shut his eyes so that he wouldn't notice the floundering legs, and only stopped when he began to feel a mild, dull pain like he had never felt before.



### Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.menu import MDDropdownMenu

KV = """
MDScreen:

 MDRaisedButton:
 id: button
 text: "PRESS ME"
 pos_hint: {"center_x": .5, "center_y": .5}
 on_release: app.menu.open()
"""

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)
 menu_items = [
 {
 "text": f"Item {i}",
 "viewclass": "OneLineListItem",
 "on_release": lambda x=f"Item {i}": self.menu_callback(x),
 } for i in range(5)
]

```

(continues on next page)

(continued from previous page)

```

self.menu = MDDropdownMenu(
 caller=self.screen.ids.button,
 items=menu_items,
 width_mult=4,
)

def menu_callback(self, text_item):
 print(text_item)

def build(self):
 return self.screen

Test().run()

```

**Warning:** Do not create the `MDDropdownMenu` object when you open the menu window. Because on a mobile device this one will be very slow!

## Wrong

```
menu = MDDropdownMenu(caller=self.screen.ids.button, items=menu_items)
menu.open()
```

## Customization of menu item

Menu items are created in the same way as items for the `RecycleView` class.

```

from kivy.lang import Builder
from kivy.metrics import dp
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBBoxLayout
from kivymd.uix.list import IRightBodyTouch, OneLineAvatarIconListItem
from kivymd.uix.menu import MDDropdownMenu

KV = '''
<RightContentCls>
 disabled: True
 adaptive_size: True
 pos_hint: {"center_y": .5}

 MDIconButton:
 icon: root.icon
 user_font_size: "16sp"
 md_bg_color_disabled: 0, 0, 0, 0

```

(continues on next page)

(continued from previous page)

```

MDLabel:
 text: root.text
 font_style: "Caption"
 adaptive_size: True
 pos_hint: {"center_y": .5}

<Item>

 IconLeftWidget:
 icon: root.left_icon

 RightContentCls:
 id: container
 icon: root.right_icon
 text: root.right_text

MDScreen:

 MDRaisedButton:
 id: button
 text: "PRESS ME"
 pos_hint: {"center_x": .5, "center_y": .5}
 on_release: app.menu.open()
 ...

class RightContentCls(IRightBodyTouch, MDBBoxLayout):
 icon = StringProperty()
 text = StringProperty()

class Item(OneLineAvatarIconListItem):
 left_icon = StringProperty()
 right_icon = StringProperty()
 right_text = StringProperty()

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)
 menu_items = [
 {
 "text": f"Item {i}",
 "right_text": f"R+{i}",
 "right_icon": "apple-keyboard-command",
 "left_icon": "git",
 "viewclass": "Item",
 "height": dp(54),

```

(continues on next page)

(continued from previous page)

```

 "on_release": lambda x=f"Item {i}": self.menu_callback(x),
 } for i in range(5)
]
self.menu = MDDropdownMenu(
 caller=self.screen.ids.button,
 items=menu_items,
 width_mult=4,
)

def menu_callback(self, text_item):
 print(text_item)

def build(self):
 return self.screen

Test().run()

```

## Header Menu

```

from kivy.lang import Builder
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.menu import MDDropdownMenu
from kivymd.uix.boxlayout import MDBBoxLayout

KV = '''
<MenuHeader>
 orientation: "vertical"
 adaptive_size: True
 padding: "4dp"

 MDBBoxLayout:
 spacing: "12dp"
 adaptive_size: True

 MDIconButton:
 icon: "gesture-tap-button"
 pos_hint: {"center_y": .5}

 MDLabel:
 text: "Actions"
 adaptive_size: True
 pos_hint: {"center_y": .5}

MDScreen:

```

(continues on next page)

(continued from previous page)

```
MDRaisedButton:
 id: button
 text: "PRESS ME"
 pos_hint: {"center_x": .5, "center_y": .5}
 on_release: app.menu.open()
...

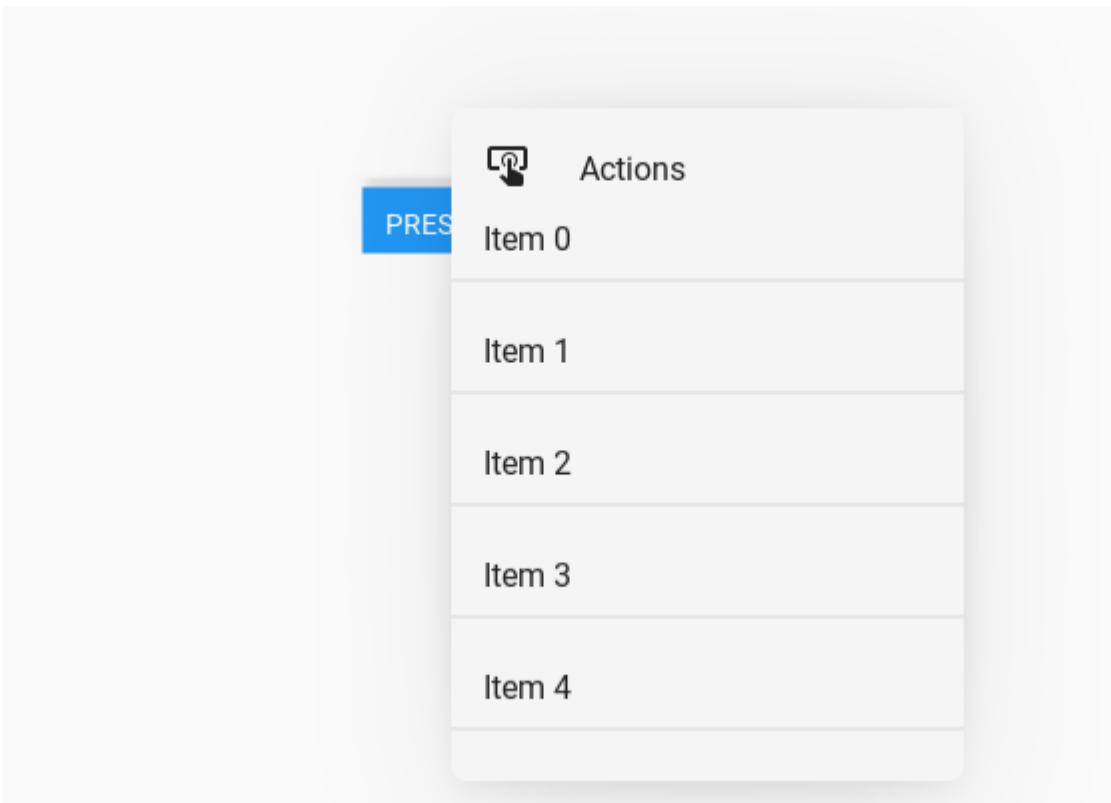
class MenuHeader(MDBoxLayout):
 "An instance of the class that will be added to the menu header."

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)
 menu_items = [
 {
 "text": f"Item {i}",
 "viewclass": "OneLineListItem",
 "height": dp(56),
 "on_release": lambda x=f"Item {i}": self.menu_callback(x),
 } for i in range(5)
]
 self.menu = MDDropdownMenu(
 header_cls=MenuHeader(),
 caller=self.screen.ids.button,
 items=menu_items,
 width_mult=4,
)

 def menu_callback(self, text_item):
 print(text_item)

 def build(self):
 return self.screen

Test().run()
```



### Menu with MDToolbar

The `MDDropdownMenu` works well with the standard `MDToolbar`. Since the buttons on the Toolbar are created by the `MDToolbar` component, it is necessary to pass the button as an argument to the callback using `lambda x: app.callback(x)`.

---

**Note:** This example uses drop down menus for both the righthand and lefthand menus (i.e both the ‘triple bar’ and ‘triple dot’ menus) to illustrate that it is possible. A better solution for the ‘triple bar’ menu would probably have been `MDDropDownMenu`.

---

```
from kivy.lang import Builder
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.menu import MDDropdownMenu
from kivymd.uix.snackbar import Snackbar

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "MDToolbar"
 left_action_items: [["menu", lambda x: app.callback(x)]]
 right_action_items: [["dots-vertical", lambda x: app.callback(x)]"]
'''
```

(continues on next page)

(continued from previous page)

```

MDLabel:
 text: "Content"
 halign: "center"
...

class Test(MDApp):
 def build(self):
 menu_items = [
 {
 "viewclass": "OneLineListItem",
 "text": f"Item {i}",
 "height": dp(56),
 "on_release": lambda x=f"Item {i}": self.menu_callback(x),
 } for i in range(5)
]
 self.menu = MDDropdownMenu(
 items=menu_items,
 width_mult=4,
)
 return Builder.load_string(KV)

 def callback(self, button):
 self.menu.caller = button
 self.menu.open()

 def menu_callback(self, text_item):
 self.menu.dismiss()
 Snackbar(text=text_item).open()

Test().run()

```

## Position menu

### Bottom position

See also:

*position*

```

from kivy.lang import Builder
from kivy.metrics import dp
from kivy.properties import StringProperty

from kivymd.uix.list import OneLineIconListItem
from kivymd.app import MDApp
from kivymd.uix.menu import MDDropdownMenu

```

(continues on next page)

(continued from previous page)

```

KV = """
<IconListItem>

 IconLeftWidget:
 icon: root.icon

MDScreen

 MDTextField:
 id: field
 pos_hint: {'center_x': .5, 'center_y': .6}
 size_hint_x: None
 width: "200dp"
 hint_text: "Password"
 on_focus: if self.focus: app.menu.open()
 ...

class IconListItem(OneLineIconListItem):
 icon = StringProperty()

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)
 menu_items = [
 {
 "viewclass": "IconListItem",
 "icon": "git",
 "height": dp(56),
 "text": f"Item {i}",
 "on_release": lambda x=f"Item {i}": self.set_item(x),
 } for i in range(5)]
 self.menu = MDDropdownMenu(
 caller=self.screen.ids.field,
 items=menu_items,
 position="bottom",
 width_mult=4,
)

 def set_item(self, text_item):
 self.screen.ids.field.text = text_item
 self.menu.dismiss()

 def build(self):
 return self.screen

Test().run()
"""

```

## Center position

```
from kivy.lang import Builder
from kivy.metrics import dp
from kivy.properties import StringProperty

from kivymd.uix.list import OneLineIconListItem
from kivymd.app import MDApp
from kivymd.uix.menu import MDDropdownMenu

KV = '''
<IconListItem>

 IconLeftWidget:
 icon: root.icon

MDScreen

 MDDropDownItem:
 id: drop_item
 pos_hint: {'center_x': .5, 'center_y': .5}
 text: 'Item 0'
 on_release: app.menu.open()
'''

class IconListItem(OneLineIconListItem):
 icon = StringProperty()

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)
 menu_items = [
 {
 "viewclass": "IconListItem",
 "icon": "git",
 "text": f"Item {i}",
 "height": dp(56),
 "on_release": lambda x=f"Item {i}": self.set_item(x),
 } for i in range(5)
]
 self.menu = MDDropdownMenu(
 caller=self.screen.ids.drop_item,
 items=menu_items,
 position="center",
 width_mult=4,
)
 self.menu.bind()
```

(continues on next page)

(continued from previous page)

```

def set_item(self, text_item):
 self.screen.ids.drop_item.set_item(text_item)
 self.menu.dismiss()

def build(self):
 return self.screen

Test().run()

```

**API - kivymd.uix.menu****class** kivymd.uix.menu.MDDropdownMenu(\*\*kwargs)**Events***on\_release* The method that will be called when you click menu items.**header\_cls**An instance of the class (*Kivy* or *KivyMD* widget) that will be added to the menu header.

New in version 0.104.2.

*header\_cls* is a [ObjectProperty](#) and defaults to *None*.**items**See [data](#).*items* is a [ListProperty](#) and defaults to *[]*.**width\_mult**

This number multiplied by the standard increment (56dp on mobile, 64dp on desktop, determines the width of the menu items.

If the resulting number were to be too big for the application Window, the multiplier will be adjusted for the biggest possible one.

*width\_mult* is a [NumericProperty](#) and defaults to *1*.**max\_height**

The menu will grow no bigger than this number. Set to 0 for no limit.

*max\_height* is a [NumericProperty](#) and defaults to *0*.**border\_margin**

Margin between Window border and menu.

*border\_margin* is a [NumericProperty](#) and defaults to *4dp*.**ver\_growth**Where the menu will grow vertically to when opening. Set to None to let the widget pick for you. Available options are: ‘*up*’, ‘*down*’.*ver\_growth* is a [OptionProperty](#) and defaults to *None*.

**hor\_growth**

Where the menu will grow horizontally to when opening. Set to None to let the widget pick for you. Available options are: ‘left’, ‘right’.

*hor\_growth* is a `OptionProperty` and defaults to `None`.

**background\_color**

Color of the background of the menu.

*background\_color* is a `ColorProperty` and defaults to `None`.

**opening\_transition**

Type of animation for opening a menu window.

*opening\_transition* is a `StringProperty` and defaults to ‘`out_cubic`’.

**opening\_time**

Menu window opening animation time and you can set it to 0 if you don’t want animation of menu opening.

*opening\_time* is a `NumericProperty` and defaults to 0.2.

**caller**

The widget object that caller the menu window.

*caller* is a `ObjectProperty` and defaults to `None`.

**position**

Menu window position relative to parent element. Available options are: ‘auto’, ‘center’, ‘bottom’.

*position* is a `OptionProperty` and defaults to ‘`auto`’.

**radius**

Menu radius.

*radius* is a `ListProperty` and defaults to ‘`[dp(7),]`’.

**check\_position\_caller(self, instance, width, height)**

**set\_menu\_properties(self, interval=0)**

Sets the size and position for the menu window.

**open(self)**

Animate the opening of a menu window.

**on\_header\_cls(self, instance, value)**

**on\_touch\_down(self, touch)**

Receive a touch down event.

**Parameters**

**touch: MotionEvent class** Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

**Returns** bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

**on\_touch\_move(self, touch)**

Receive a touch move event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

**on\_touch\_up(self, touch)**

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

**on\_dismiss(self)**

Called when the menu is closed.

**dismiss(self)**

Closes the menu.

### 2.3.3 MDSwiper

#### Usage

**MDSwiper:**

**MDSwiperItem:**

**MDSwiperItem:**

**MDSwiperItem:**

#### Example

```
from kivymd.app import MDApp
from kivy.lang.builder import Builder

kv = '''
<MySwiper@MDSwiperItem>

 FitImage:
 source: "guitar.png"
 radius: [20,]

MDScreen:

 MDToolbar:
 id: toolbar
 title: "MDSwiper"
 elevation: 10
 pos_hint: {"top": 1}

 MDSwiper:
 size_hint_y: None
 height: root.height - toolbar.height - dp(40)
 y: root.height - self.height - toolbar.height - dp(20)

 MySwiper:

 MySwiper:
```

(continues on next page)

(continued from previous page)

```
MySwiper:
MySwiper:
MySwiper:
...

class Main(MDApp):
 def build(self):
 return Builder.load_string(kv)

Main().run()
```

**Warning:** The width of *MDSwiperItem* is adjusted automatically. Consider changing that by `width_mult`.

**Warning:** The width of *MDSwiper* is automatically adjusted according to the width of the window.

**MDSwiper provides the following events for use:**

```
__events__ = (
 "on_swipe",
 "on_pre_swipe",
 "on_overswipe_right",
 "on_overswipe_left",
 "on_swipe_left",
 "on_swipe_right"
)
```

```
MDSwiper:
on_swipe: print("on_swipe")
on_pre_swipe: print("on_pre_swipe")
on_overswipe_right: print("on_overswipe_right")
on_overswipe_left: print("on_overswipe_left")
on_swipe_left: print("on_swipe_left")
on_swipe_right: print("on_swipe_right")
```

## Example

```
from kivy.lang.builder import Builder

from kivymd.app import MDApp

kv = '''
<MagicButton@MagicBehavior+MDIconButton>

<MySwiper@MDSwiperItem>

RelativeLayout:

 FitImage:
 source: "guitar.png"
 radius: [20,]

 MDBBoxLayout:
 adaptive_height: True
 spacing: "12dp"

 MagicButton:
 id: icon
 icon: "weather-sunny"
 user_font_size: "56sp"
 opposite_colors: True

 MDLabel:
 text: "MDLabel"
 font_style: "H5"
 size_hint_y: None
 height: self.texture_size[1]
 pos_hint: {"center_y": .5}
 opposite_colors: True

MDScreen:

 MDToolbar:
 id: toolbar
 title: "MDSwiper"
 elevation: 10
 pos_hint: {"top": 1}

 MDSwiper:
 size_hint_y: None
 height: root.height - toolbar.height - dp(40)
 y: root.height - self.height - toolbar.height - dp(20)
 on_swipe: self.get_current_item().ids.icon.shake()

 MySwiper:
```

(continues on next page)

(continued from previous page)

```
MySwiper:
MySwiper:
MySwiper:
MySwiper:
...

class Main(MDApp):
 def build(self):
 return Builder.load_string(kv)

Main().run()
```

## How to automatically switch a SwiperItem?

Use method `set_current` which takes the index of `MDSwiperItem` as argument.

### Example

```
MDSwiper:
 id: swiper

 MDSwiperItem: # First widget with index 0

 MDSwiperItem: # Second widget with index 1

MDRaisedButton:
 text: "Go to Second"
 on_release: swiper.set_current(1)
```

## API - kivymd.uix.swiper

`class kivymd.uix.swiper.MDSwiperItem(**kwargs)`  
`MDSwiperItem` is a BoxLayout but it's size is adjusted automatically.

`class kivymd.uix.swiper.MDSwiper(**kwargs)`  
ScrollView class. See module documentation for more information.

### Events

- `on_scroll_start` Generic event fired when scrolling starts from touch.
- `on_scroll_move` Generic event fired when scrolling move from touch.
- `on_scroll_stop` Generic event fired when scrolling stops from touch.

Changed in version 1.9.0: `on_scroll_start`, `on_scroll_move` and `on_scroll_stop` events are now dispatched when scrolling to handle nested ScrollViews.

Changed in version 1.7.0: `auto_scroll`, `scroll_friction`, `scroll_moves`, `scroll_stoptime`' has been deprecated, use `:attr: `effect_cls`` instead.

### **items\_spacing**

The space between each `MDSwiperItem`.

`items_spacing` is an `NumericProperty` and defaults to `20dp`.

### **transition\_duration**

Duration of switching between `MDSwiperItem`.

`transition_duration` is an `NumericProperty` and defaults to `0.2`.

### **size\_duration**

Duration of changing the size of `MDSwiperItem`.

`transition_duration` is an `NumericProperty` and defaults to `0.2`.

### **size\_transition**

The type of animation used for changing the size of `MDSwiperItem`.

`size_transition` is an `StringProperty` and defaults to `out_quad`.

### **swipe\_transition**

The type of animation used for swiping.

`swipe_transition` is an `StringProperty` and defaults to `out_quad`.

### **swipe\_distance**

Distance to move before swiping the `MDSwiperItem`.

`swipe_distance` is an `NumericProperty` and defaults to `70dp`.

### **width\_mult**

This number is multiplied by `items_spacing` x2 and then subtracted from the width of window to specify the width of `MDSwiperItem`. So by decreasing the `width_mult` the width of `MDSwiperItem` increases and vice versa.

`width_mult` is an `NumericProperty` and defaults to `3`.

### **swipe\_on\_scroll**

Wheter to swipe on mouse wheel scrolling or not.

`swipe_on_scroll` is an `BooleanProperty` and defaults to `True`.

### **add\_widget(self, widget, index=0)**

Add a new widget as a child of this widget.

#### **Parameters**

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

**remove\_widget(self, widget)**

Remove a widget from the children of this widget.

**Parameters**

**widget: Widget** Widget to remove from our children list.

```
>>> from kivy.uix.button import Button
>>> root = Widget()
>>> button = Button()
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

**set\_current(self, index)**

Switch to given *MDSwiperItem* index.

**get\_current\_index(self)**

Returns the current *MDSwiperItem* index.

**get\_current\_item(self)**

Returns the current *MDSwiperItem* instance.

**get\_items(self)**

Returns the list of *MDSwiperItem* children.

---

**Note:** Use *get\_items()* to get the list of children instead of *MDSwiper.children*.

---

**on\_swipe(self)****on\_preSwipe(self)****on\_overswipe\_right(self)****on\_overswipe\_left(self)****on\_swipe\_left(self)****on\_swipe\_right(self)****swipe\_left(self)****swipe\_right(self)****on\_scroll\_start(self, touch, check\_children=True)****on\_touch\_down(self, touch)**

Receive a touch down event.

**Parameters**

**touch: MotionEvent class** Touch received. The touch is in parent coordinates. See [relativelayout](#) for a discussion on coordinate systems.

**Returns** bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

#### `on_touch_up(self, touch)`

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

### 2.3.4 DataTables

#### See also:

Material Design spec, DataTables

**Data tables display sets of data across rows and columns.**

|                                     |         |                            |
|-------------------------------------|---------|----------------------------|
| <input type="checkbox"/>            | Online  | Astrid: NE shared mail     |
| <input checked="" type="checkbox"/> | Offline | Cosmo: prod shared account |
| <input checked="" type="checkbox"/> | Online  | Phoenix: prod shared I     |
| <input type="checkbox"/>            | Online  | Sirius: prod shared ac     |

#### Warnings

**Warning:** Data tables are still far from perfect. The class is in constant change, because of optimizations and bug fixes.

If you find a bug or have an improvement you want to share, take some time and share your discoveries with us over the main git repo.

Any help is well appreciated.

**Warning:** In versions prior to Kivy 2.1-dev0 exists an error in which is the table has only one row in the current page, the table will only render one column instead of the whole row.

**Note:** MDDDataTable allows developers to sort the data provided by column. This happens thanks to the use of an external function that you can bind while you're defining the table columns.

Be aware that the sorting function must return a 2 value list in the format of:

*[Index, Sorted\_Row\_Data]*

This is because the index list is needed to allow MDDDataTable to keep track of the selected rows. and, after the data is sorted, update the row checkboxes.

---

## API - kivymd.uix.datatables

```
class kivymd.uix.datatables.MDDDataTable(**kwargs)
```

### Events

***on\_row\_press*** Called when a table row is clicked.

***on\_check\_press*** Called when the check box in the table row is checked.

### Use events as follows

```
from kivy.metrics import dp
from kivy.uix.anchorlayout import AnchorLayout
from kivy.lang import Builder
from kivy.logger import Logger

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable

kv = '''
BoxLayout:
 orientation: "vertical"
 BoxLayout:
 id:button_tab
 size_hint_y:None
 height: dp(48)

 MDFlatButton:
 text: "Hello <3"
 on_release:
 app.update_row_data()

 BoxLayout:
 id:body

 ...

class Example(MDApp):
 def build(self):
 self.data_tables = MDDDataTable(
 # MDDDataTable allows the use of size_hint
 size_hint=(0.8, 0.7),
 use_pagination=True,
 check=True,
```

(continues on next page)

(continued from previous page)

```

column_data=[

 ("No.", dp(30)),

 ("Status", dp(30)),

 ("Signal Name", dp(60), self.sort_on_signal),

 ("Severity", dp(30)),

 ("Stage", dp(30)),

 ("Schedule", dp(30), self.sort_on_schedule),

 ("Team Lead", dp(30), self.sort_on_team)

],

row_data=[

 ("1", ("alert", [255 / 256, 165 / 256, 0, 1], "No Signal"),

 "Astrid: NE shared managed", "Medium", "Triaged", "0:33",

 "Chase Nguyen"),

 ("2", ("alert-circle", [1, 0, 0, 1], "Offline"),

 "Cosmo: prod shared ares", "Huge", "Triaged", "0:39",

 "Brie Furman"),

 ("3", (

 "checkbox-marked-circle",

 [39 / 256, 174 / 256, 96 / 256, 1],

 "Online"), "Phoenix: prod shared lyra-lists", "Minor",

 "Not Triaged", "3:12", "Jeremy lake"),

 ("4", (

 "checkbox-marked-circle",

 [39 / 256, 174 / 256, 96 / 256, 1],

 "Online"), "Sirius: NW prod shared locations",

 "Negligible",

 "Triaged", "13:18", "Angelica Howards"),

 ("5", (

 "checkbox-marked-circle",

 [39 / 256, 174 / 256, 96 / 256, 1],

 "Online"), "Sirius: prod independent account",

 "Negligible",

 "Triaged", "22:06", "Diane Okuma"),

],

 sorted_on="Schedule",

 sorted_order="ASC",

 elevation=2

)

self.data_tables.bind(on_row_press=self.on_row_press)

self.data_tables.bind(on_check_press=self.on_check_press)

root = Builder.load_string(kv)

root.ids.body.add_widget(self.data_tables)

return root

def update_row_data(self, *dt):

 self.data_tables.row_data = [

 (

 "21",

 ("alert", [255 / 256, 165 / 256, 0, 1], "No Signal"),

 "Astrid: NE shared managed",

 "Medium",

```

(continues on next page)

(continued from previous page)

```

 "Triaged",
 "0:33",
 "Chase Nguyen"
),
("32", ("alert-circle", [1, 0, 0, 1], "Offline"),
"Cosmo: prod shared ares", "Huge", "Triaged", "0:39",
"Brie Furman"),
("43", (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online"), "Phoenix: prod shared lyra-lists", "Minor",
"Not Triaged", "3:12", "Jeremy lake"),
("54", (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online"), "Sirius: NW prod shared locations",
"Negligible",
"Triaged", "13:18", "Angelica Howards"),
("85", (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online"), "Sirius: prod independent account",
"Negligible",
"Triaged", "22:06", "Diane Okuma"),
("85", (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online"), "Sirius: prod independent account",
"Negligible",
"Triaged", "22:06", "John Sakura"),
]

def on_row_press(self, instance_table, instance_row):
 "Called when a table row is clicked."
 print(instance_table, instance_row)

def on_check_press(self, instance_table, current_row):
 "Called when the check box in the table row is checked."
 print(instance_table, current_row)

Sorting Methods:
Since the # 914 Pull request, the sorting method requires you to sort
out the indexes of each data value for the support of selections

The most common method to do this is with the use of the built-in function
zip and enumerate, see the example below for more info.

the result given by these functions must be a list in the format of
[Indexes, Sorted_Row_Data]

```

(continues on next page)

(continued from previous page)

```

def sort_on_signal(self, data):
 return zip(
 *sorted(
 enumerate(data),
 key=lambda l: l[1][2]
)
)

def sort_on_schedule(self, data):
 return zip(
 *sorted(
 enumerate(data),
 key=lambda l: sum(
 [int(l[1][-2].split(":")[0])*60,
 int(l[1][-2].split(":")[1])]
)
)
)

def sort_on_team(self, data):
 return zip(
 *sorted(
 enumerate(data),
 key=lambda l: l[1][-1]
)
)

```

Example().run()

**column\_data**

Data for header columns.

```

from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable
from kivy.uix.anchorlayout import AnchorLayout

class Example(MDApp):
 def build(self):
 layout = AnchorLayout()
 self.data_tables = MDDDataTable(
 size_hint=(0.7, 0.6),
 use_pagination=True,
 check=True,

 # name column, width column, sorting function column(optional)
 column_data=[("No.", dp(30)),

```

(continues on next page)

(continued from previous page)

```

 ("Status", dp(30)),
 ("Signal Name", dp(60)),
 ("Severity", dp(30)),
 ("Stage", dp(30)),
 ("Schedule", dp(30), lambda *args: print("Sorted using Schedule
←")),
 ("Team Lead", dp(30)),
],
)
layout.add_widget(self.data_tables)
return layout

```

`Example().run()`

| Column 1 | Column 2 | Column 3 | Column 4 | Column 5 | Column 6 |
|----------|----------|----------|----------|----------|----------|
|          |          |          |          |          |          |

`column_data` is an `ListProperty` and defaults to `[]`.

**Note:** The functions which will be called for sorting must accept a data argument and return the sorted data.

Incoming data format will be similar to the provided `row_data` except that it'll be all list instead of tuple like below. Any icon provided initially will also be there in this data so handle accordingly.

```
[
 ['1', ['icon', 'No Signal'], 'Astrid: NE shared managed', 'Medium', 'Triaged
←', '0:33', 'Chase Nguyen'],
 ['2', 'Offline', 'Cosmo: prod shared ares', 'Huge', 'Triaged', '0:39',
←'Brie Furman'],
 ['3', 'Online', 'Phoenix: prod shared lyra-lists', 'Minor', 'Not Triaged',
←'3:12', 'Jeremy lake'],
 ['4', 'Online', 'Sirius: NW prod shared locations', 'Negligible', 'Triaged',
←'13:18', 'Angelica Howards'],
 ['5', 'Online', 'Sirius: prod independent account', 'Negligible', 'Triaged',
←'22:06', 'Diane Okuma']
]
```

You must sort inner lists in ascending order and return the sorted data in the same format.

#### **row\_data**

Data for rows. To add icon in addition to a row data, include a tuple with This property stores the row data used to display each row in the DataTable To show an icon inside a column in a row, use the folowing format in the row's columns.

Format:

(“MDicon-name”, [icon color in rgba], “Column Value”)

Example:

For a more complex example see below.

```
from kivy.metrics import dp
from kivy.uix.anchorlayout import AnchorLayout

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable

class Example(MDApp):
 def build(self):
 layout = AnchorLayout()
 data_tables = MDDDataTable(
 size_hint=(0.9, 0.6),
 column_data=[
 ("Column 1", dp(20)),
 ("Column 2", dp(30)),
 ("Column 3", dp(50), self.sort_on_col_3),
 ("Column 4", dp(30)),
 ("Column 5", dp(30)),
 ("Column 6", dp(30)),
 ("Column 7", dp(30), self.sort_on_col_2),
],
 row_data=[
 # The number of elements must match the length
 # of the `column_data` list.
 (
 "1",
 ("alert", [255 / 256, 165 / 256, 0, 1], "No Signal"),
 "Astrid: NE shared managed",
 "Medium",
 "Triaged",
 "0:33",
 "Chase Nguyen",
),
 (
 "2",
 ("alert-circle", [1, 0, 0, 1], "Offline"),
 "Cosmo: prod shared ares",
 "Huge",
 "Triaged",
 "0:39",
)
]
)
 layout.add_widget(data_tables)
 return layout
```

(continues on next page)

(continued from previous page)

```

 "Brie Furman",
),
 (
 "3",
 (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online",
),
 "Phoenix: prod shared lyra-lists",
 "Minor",
 "Not Triaged",
 "3:12",
 "Jeremy lake",
),
 (
 "4",
 (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online",
),
 "Sirius: NW prod shared locations",
 "Negligible",
 "Triaged",
 "13:18",
 "Angelica Howards",
),
 (
 "5",
 (
 "checkbox-marked-circle",
 [39 / 256, 174 / 256, 96 / 256, 1],
 "Online",
),
 "Sirius: prod independent account",
 "Negligible",
 "Triaged",
 "22:06",
 "Diane Okuma",
),
),
],
)
layout.add_widget(data_tables)
return layout

def sort_on_col_3(self, data):
 return zip(
 *sorted(
 enumerate(data),
 key=lambda l: l[1][3]
)
)

```

(continues on next page)

(continued from previous page)

```

)

def sort_on_col_2(self, data):
 return zip(
 *sorted(
 enumerate(data),
 key=lambda l: l[1] [-1]
)
)

Example().run()

```

| Column 1 | Column 2    | Column 3                         | Column 4   | Column 5    | Column 6 | Column 7         |
|----------|-------------|----------------------------------|------------|-------------|----------|------------------|
| 1        | ⚠ No Signal | Astrid: NE shared managed        | Medium     | Triaged     | 0:33     | Chase Nguyen     |
| 2        | ❗ Offline   | Cosmo: prod shared ares          | Huge       | Triaged     | 0:39     | Brie Furman      |
| 3        | ✓ Online    | Phoenix: prod shared lyra-lists  | Minor      | Not Triaged | 3:12     | Jeremy lake      |
| 4        | ✓ Online    | Sirius: NW prod shared locations | Negligible | Triaged     | 13:18    | Angelica Howards |
| 5        | ✓ Online    | Sirius: prod independent account | Negligible | Triaged     | 22:06    | Diane Okuma      |

*row\_data* is an `ListProperty` and defaults to `[]`.

#### **sorted\_on**

Column name upon which the data is already sorted.

If the table data is showing an already sorted data then this can be used to indicate upon which column the data is sorted.

*sorted\_on* is an `StringProperty` and defaults to ''.

#### **sorted\_order**

Order of already sorted data. Must be one of 'ASC' for ascending or 'DSC' for descending order.

*sorted\_order* is an `OptionProperty` and defaults to 'ASC'.

#### **check**

Use or not use checkboxes for rows.

*check* is an `BooleanProperty` and defaults to `False`.

#### **use\_pagination**

Use page pagination for table or not.

```

from kivy.metrics import dp
from kivy.uix.anchorlayout import AnchorLayout

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable

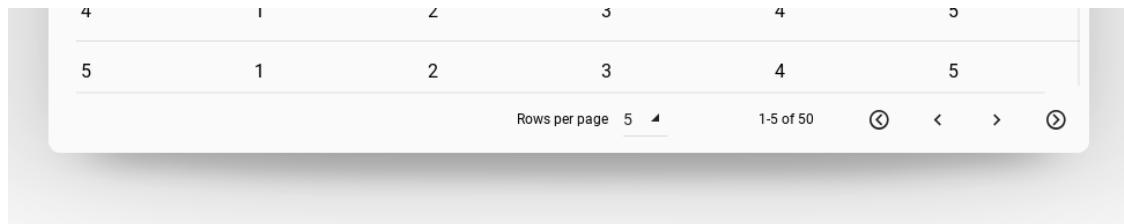
```

(continues on next page)

(continued from previous page)

```
class Example(MDApp):
 def build(self):
 layout = AnchorLayout()
 data_tables = MDDataTable(
 size_hint=(0.9, 0.6),
 use_pagination=True,
 column_data=[
 ("No.", dp(30)),
 ("Column 1", dp(30)),
 ("Column 2", dp(30)),
 ("Column 3", dp(30)),
 ("Column 4", dp(30)),
 ("Column 5", dp(30)),
],
 row_data=[
 (f"{i + 1}", "1", "2", "3", "4", "5") for i in range(50)
],
)
 layout.add_widget(data_tables)
 return layout
```

```
Example().run()
```



`use_pagination` is an `BooleanProperty` and defaults to `False`.

#### elevation

Table elevation.

`elevation` is an `NumericProperty` and defaults to `8`.

#### rows\_num

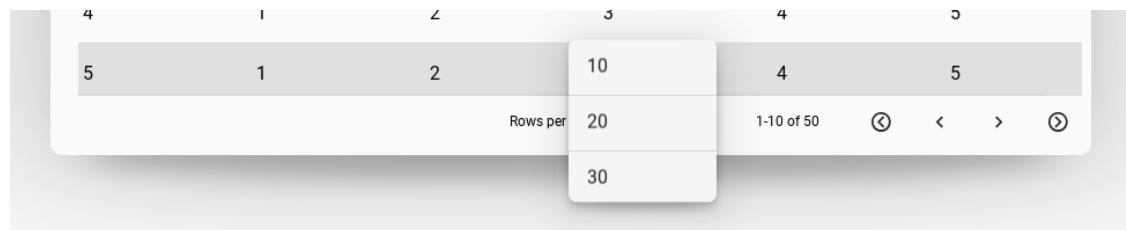
The number of rows displayed on one page of the table.

`rows_num` is an `NumericProperty` and defaults to `10`.

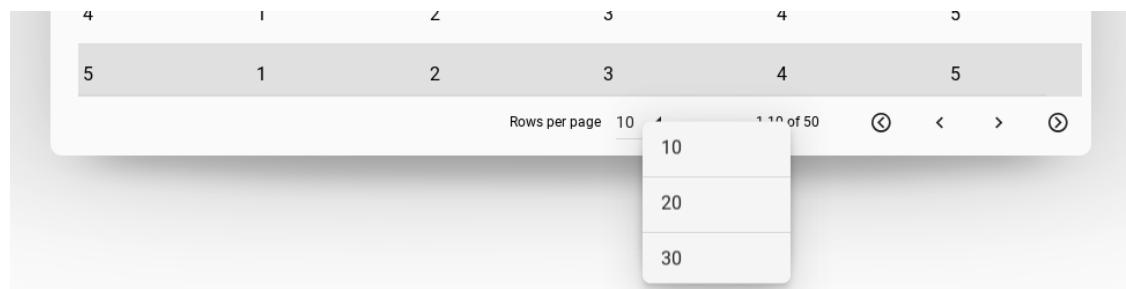
#### pagination\_menu\_pos

Menu position for selecting the number of displayed rows. Available options are ‘center’, ‘auto’.

### Center



### Auto

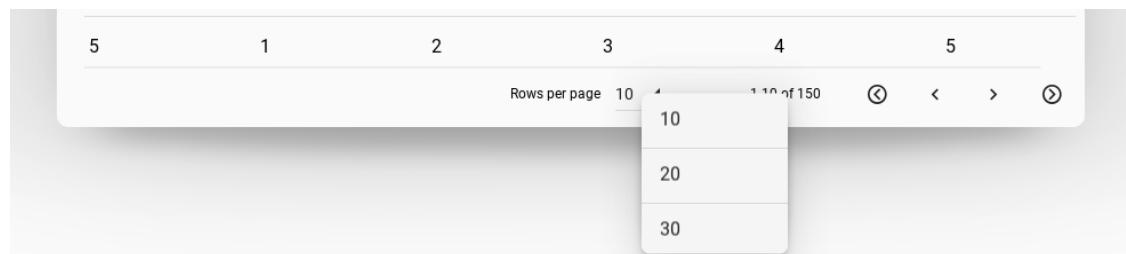


`pagination_menu_pos` is an `OptionProperty` and defaults to ‘center’.

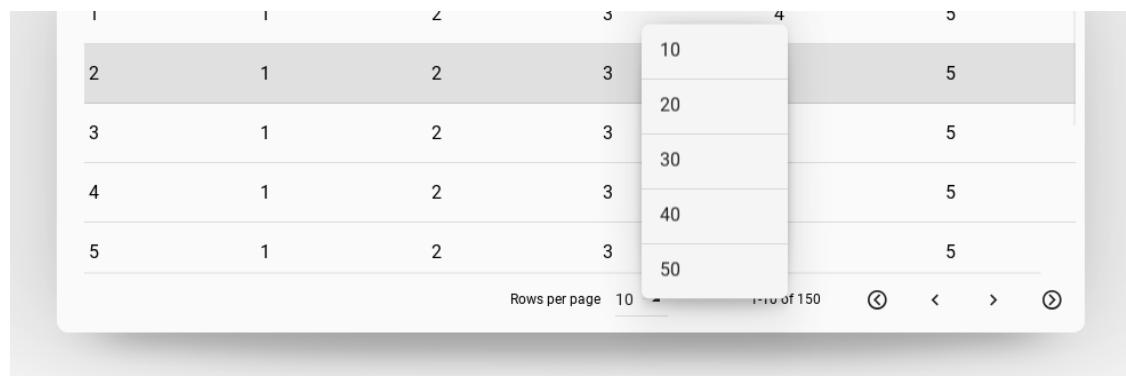
#### `pagination_menu_height`

Menu height for selecting the number of displayed rows.

**140dp**



**240dp**



*pagination\_menu\_height* is an NumericProperty and defaults to ‘140dp’.

#### background\_color

Background color in the format (r, g, b, a). See [background\\_color](#).

```
from kivy.metrics import dp
from kivy.uix.anchorlayout import AnchorLayout

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable

class Example(MDApp):
 def build(self):
 layout = AnchorLayout()
 data_tables = MDDDataTable(
 size_hint=(0.9, 0.6),
 use_pagination=True,
 column_data=[
 ("No.", dp(30)),
 ("Column 1", dp(30)),
 ("[color=#52251B]Column 2[/color]", dp(30)),
 ("Column 3", dp(30)),
 ("[size=24][color=#C042B8]Column 4[/color][/size]", dp(30)),
 ("Column 5", dp(30)),
],
 row_data=[
 (
 f"{i + 1}",
 "[color=#297B50]1[/color]",
 "[color=#C552A1]2[/color]",
 "[color=#6C9331]3[/color]",
 "4",
 "5",
)
 for i in range(50)
],
)
 layout.add_widget(data_tables)
 return layout

Example().run()
```

| No. | Column 1 | Column 2 | Column 3 | Column 4 | Column 5 |
|-----|----------|----------|----------|----------|----------|
| 1   | 1        | 2        | 3        | 4        | 5        |
| 2   | 1        | 2        | 3        | 4        | 5        |
| 3   | 1        | 2        | 3        | 4        | 5        |
| 4   | 1        | 2        | 3        | 4        | 5        |
| 5   | 1        | 2        | 3        | 4        | 5        |

Rows per page: 5 | 1-5 of 50 | < >

`background_color` is a `ColorProperty` and defaults to [0, 0, 0, 0].

`update_row_data(self, instance, value)`

Called when a the widget data must be updated.

Remember that this is a heavy function. since the whole data set must be updated. you can get better results calling this metod with in a coroutine.

`on_row_press(self, *args)`

Called when a table row is clicked.

`on_check_press(self, *args)`

Called when the check box in the table row is checked.

`get_row_checks(self)`

Returns all rows that are checked.

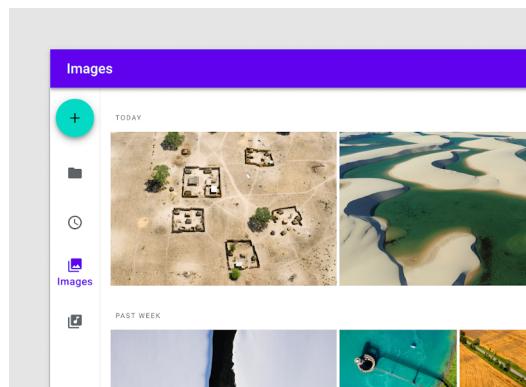
`create_pagination_menu(self, interval)`

### 2.3.5 Navigation Rail

See also:

Material Design spec, Navigation rail

Navigation rails provide ergonomic movement between primary destinations in apps.



## Usage

**MDNavigationRail:**

**MDNavigationRailItem:**

**MDNavigationRailItem:**

**MDNavigationRailItem:**

```
from kivy.factory import Factory
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
#:import get_color_from_hex kivy.utils.get_color_from_hex

<MyTile@SmartTileWithStar>
 size_hint_y: None
 height: "240dp"

```

```
MDBBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "MDNavigationRail"
 md_bg_color: rail.md_bg_color

 MDBBoxLayout:

 MDNavigationRail:
 id: rail
 md_bg_color: get_color_from_hex("#344954")
```

(continues on next page)

(continued from previous page)

```
color_normal: get_color_from_hex("#718089")
color_active: get_color_from_hex("#f3ab44")

MDNavigationRailItem:
 icon: "language-cpp"
 text: "C++"

MDNavigationRailItem:
 icon: "language-python"
 text: "Python"

MDNavigationRailItem:
 icon: "language-swift"
 text: "Swift"

MDBBoxLayout:
 padding: "24dp"

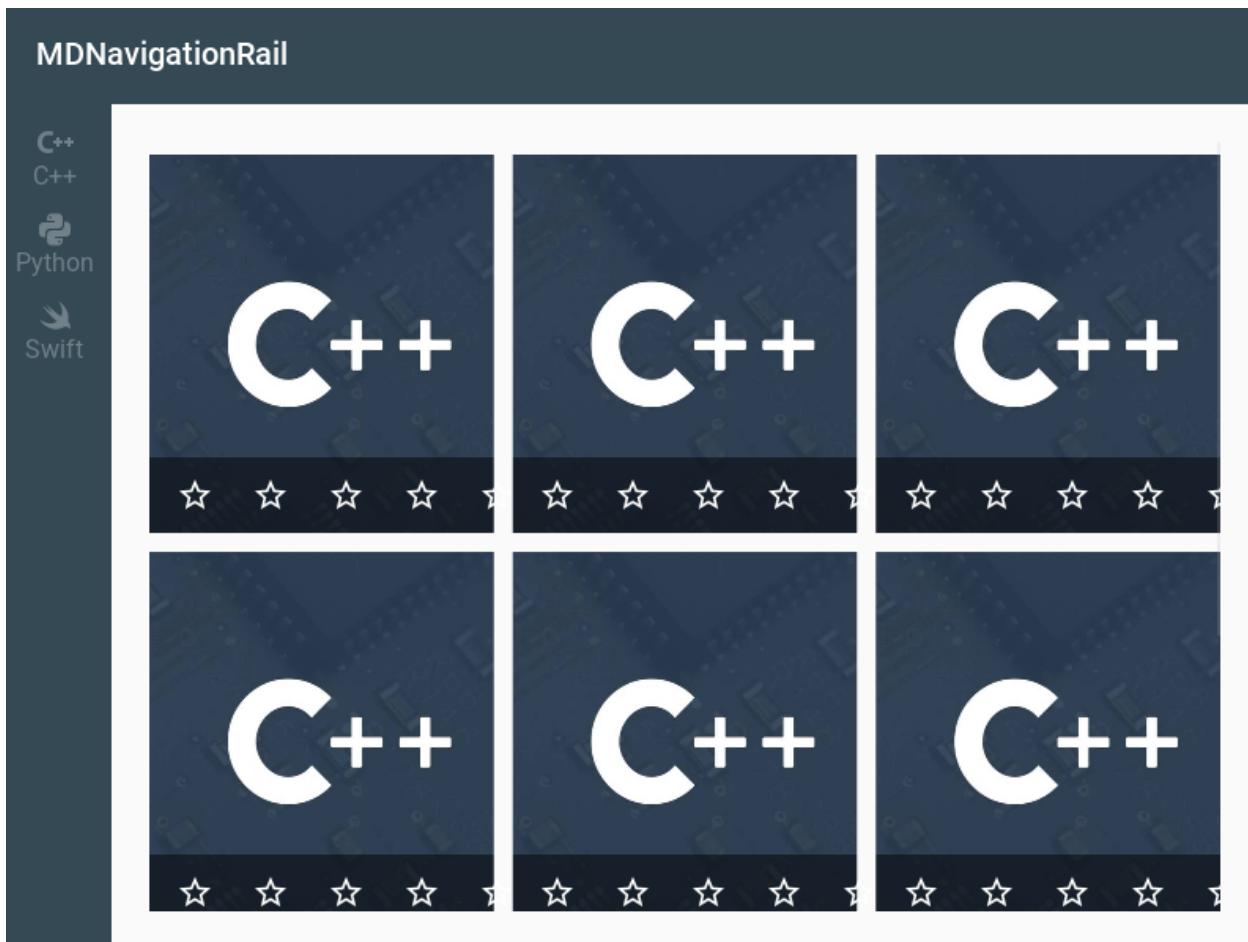
ScrollView:

 MDList:
 id: box
 cols: 3
 spacing: "12dp"
 ...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for i in range(9):
 tile = Factory.MyTile(source="cpp.png")
 tile.stars = 5
 self.root.ids.box.add_widget(tile)

Test().run()
```



#### API - `kivymd.uix.navigationrail`

```
class kivymd.uix.navigationrail.MDNavigationRail(**kwargs)
```

##### Events

**`on_item_switch`** Called when the menu item is switched.

**`on_open`** Called when a rail is opened.

**`on_close`** Called when a rail is closed.

**`on_action_button`**

##### **`use_hover_behavior`**

Whether to use the HoverBehavior effect for menu items.

```
MDNavigationRail:
 use_hover_behavior: True
 hover_bg: 0, 0, 0, .2
```

`use_hover_behavior` is an `BooleanProperty` and defaults to `False`.

**hover\_bg**

The background color for the menu item. Used when `use_hover_behavior` parameter is *True*.

**use\_resizeable**

Allows you to change the width of the rail (open/close).

```
from kivy.factory import Factory
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
#:import get_color_from_hex kivy.utils.get_color_from_hex

<MyTile@SmartTileWithStar>
 size_hint_y: None
 height: "240dp"

MDBBoxLayout:
 orientation: "vertical"

MDToolbar:
 title: "MDNavigationRail"
 md_bg_color: rail.md_bg_color
 left_action_items: [["menu", lambda x: app.rail_open()]]

MDBBoxLayout:

 MDNavigationRail:
 id: rail
 md_bg_color: get_color_from_hex("#344954")
 color_normal: get_color_from_hex("#718089")
 color_active: get_color_from_hex("#f3ab44")
 use_resizeable: True

 MDNavigationRailItem:
 icon: "language-cpp"
 text: "C++"

 MDNavigationRailItem:
 icon: "language-java"
 text: "Java"

 MDNavigationRailItem:
 icon: "language-swift"
 text: "Swift"

 MDBBoxLayout:
 padding: "24dp"

 ScrollView:
```

(continues on next page)

(continued from previous page)

```

MDList:
 id: box
 cols: 3
 spacing: "12dp"
 ...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def rail_open(self):
 if self.root.ids.rail.rail_state == "open":
 self.root.ids.rail.rail_state = "close"
 else:
 self.root.ids.rail.rail_state = "open"

 def on_start(self):
 for i in range(9):
 tile = Factory.MyTile(source="kitten.png")
 tile.stars = 5
 self.root.ids.box.add_widget(tile)

Test().run()

```

`use_resizeable` is an `BooleanProperty` and defaults to `False`.

#### `use_title`

Whether to use an additional panel at the top of the rail.

```

MDNavigationRail:
 use_resizeable: True
 use_title: True
 icon_title: "logo.png"
 text_title: "[b][color=#ffffff]Example[/color][/b]"

```

`use_title` is an `BooleanProperty` and defaults to `False`.

#### `icon_title`

Icon (name or path to `png` file) for `NavigationRailTitle` class.

`icon_title` is an `StringProperty` and defaults to ‘`menu`’.

#### `text_title`

Text title for `NavigationRailTitle` class.

`text_title` is an `StringProperty` and defaults to ‘`Rail`’.

#### `use_action_button`

Should `MDFloatingActionButton` button be used.

```
MDNavigationRail:
 use_action_button: True
 action_text_button: "COMPOSE"
 on_action_button: print(args)
```

`use_action_button` is an `BooleanProperty` and defaults to `False`.

**action\_icon\_button**

Icon of `use_action_button`.

`action_icon_button` is an `StringProperty` and defaults to '`plus`'.

**action\_text\_button**

Text of `use_action_button`.

`action_text_button` is an `StringProperty` and defaults to ''.

**action\_color\_button**

Text of `use_action_button`.

`action_color_button` is an `ColorProperty` and defaults to `None`.

**color\_normal**

Color normal of item menu.

`color_normal` is an `ColorProperty` and defaults to `None`.

**color\_active**

Color active of item menu.

`color_active` is an `ColorProperty` and defaults to `None`.

**visible**

Item label visible type. Available options are: '`Selected`', '`Persistent`', '`Unlabeled`'.

```
MDNavigationRail:
 visible: "Persistent"
```

```
MDNavigationRail:
 visible: "Selected"
```

```
MDNavigationRail:
 visible: "Unlabeled"
```

`visible` is an `OptionProperty` and defaults to '`Persistent`'.

**color\_transition**

Animation type when changing the color of a menu item.

`color_transition` is a `StringProperty` and defaults to '`linear`'.

**color\_change\_duration**

Animation duration when changing menu item color.

`color_change_duration` is a `NumericProperty` and defaults to `0.2`.

**rail\_state**

Closed or open rails.

`rail_state` is a `OptionProperty` and defaults to ‘close’.

**anim\_color\_normal(self, item)****anim\_color\_active(self, item)****item\_switch(self, instance\_item)****add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

**Parameters**

`widget: Widget` Widget to add to our list of children.

`index: int, defaults to 0` Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

`canvas: str, defaults to None` Canvas to add widget’s canvas to. Can be ‘before’, ‘after’ or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

**open(self)****close(self)****on\_rail\_state(self, instance, value)****on\_item\_switch(self, instance\_item)**

Called when the menu item is switched.

**on\_open(self)**

Called when a rail is opened.

**on\_close(self)**

Called when a rail is closed.

**on\_action\_button(self, floatingActionButton)**

Called when the `MDFloatingActionButton` is pressed.

**on\_visible(self, instance, value)****on\_use\_title(self, instance, value)****on\_use\_resizeable(self, instance, value)****on\_useActionButton(self, instance, value)****press\_floatingActionButton(self, floatingActionButton)****setActionButtonColor(self, interval)**

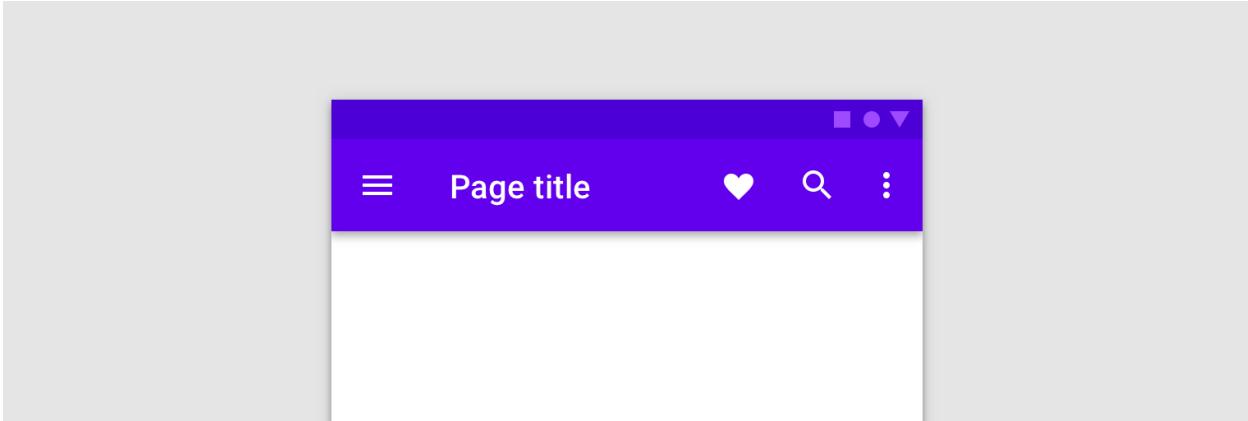
```
set_width(self, interval)
set_box_title_size(self, interval)
set_action_icon_button(self, interval)
set_action_text_button(self, interval)
set_color_menu_item(self, instance_item)
set_items_color(self, interval)
set_items_visible(self, interval)
```

### 2.3.6 Toolbar

**See also:**

[Material Design spec, App bars: top](#)

[Material Design spec, App bars: bottom](#)



KivyMD provides the following toolbar positions for use:

- *Top*
- *Bottom*

#### Top

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "MDToolbar"
```

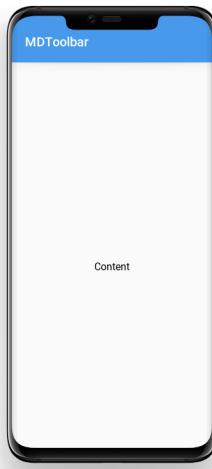
(continues on next page)

(continued from previous page)

```
MDLabel:
 text: "Content"
 halign: "center"
...

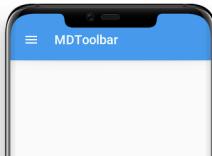
class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```



### Add left menu

```
MDToolbar:
 title: "MDToolbar"
 left_action_items: [["menu", lambda x: app.callback()]]
```



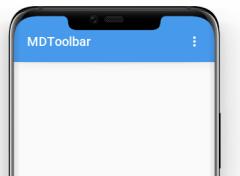
---

**Note:** The callback is optional. `left_action_items: [[{"menu"}]]` would also work for a button that does nothing.

---

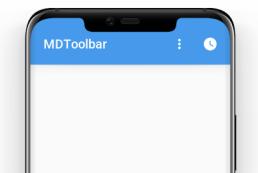
### Add right menu

```
MDToolbar:
 title: "MDToolbar"
 right_action_items: [["dots-vertical", lambda x: app.callback()]]
```



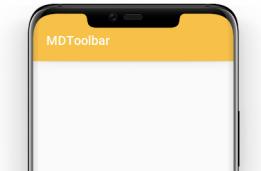
### Add two item to the right menu

```
MDToolbar:
 title: "MDToolbar"
 right_action_items: [["dots-vertical", lambda x: app.callback_1()], ["clock", lambda x: app.callback_2()]]
```



### Change toolbar color

```
MDToolbar:
 title: "MDToolbar"
 md_bg_color: app.theme_cls.accent_color
```



### Change toolbar text color

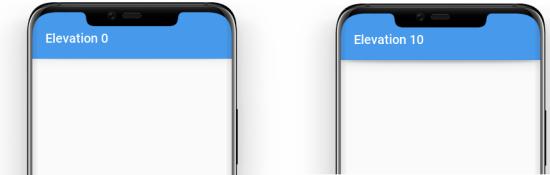
```
MDToolbar:
 title: "MDToolbar"
 specific_text_color: app.theme_cls.accent_color
```



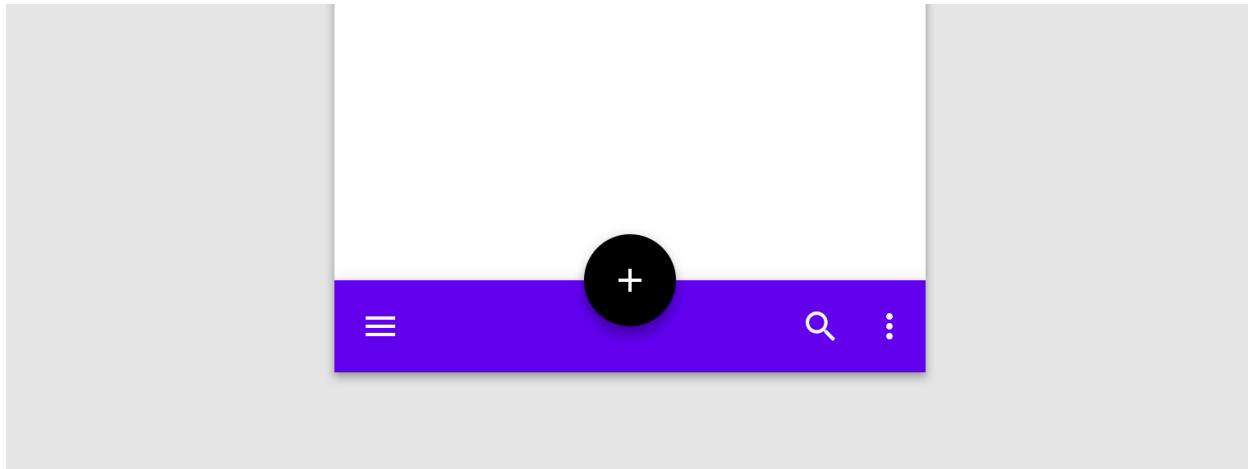
### Shadow elevation control

#### MDToolbar:

```
 title: "Elevation 10"
 elevation: 10
```



### Bottom



### Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = ''''
MDBottomAppBar:

 # Will always be at the bottom of the screen.
 MDBottomAppBar:
```

(continues on next page)

(continued from previous page)

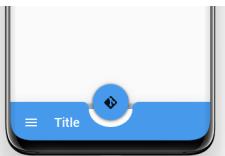
```

MDToolbar:
 title: "Title"
 icon: "git"
 type: "bottom"
 left_action_items: [["menu", lambda x: x]]
...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()

```



## Event on floating button

Event on\_action\_button:

```

MDBottomAppBar:

 MDToolbar:
 title: "Title"
 icon: "git"
 type: "bottom"
 left_action_items: [["menu", lambda x: x]]
 on_action_button: app.callback(self.icon)

```

## Floating button position

Mode:

- 'free-end'
- 'free-center'
- 'end'
- 'center'

```

MDBottomAppBar:

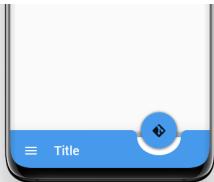
 MDToolbar:
 title: "Title"
 icon: "git"

```

(continues on next page)

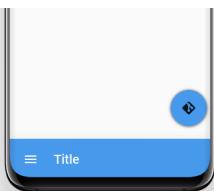
(continued from previous page)

```
type: "bottom"
left_action_items: [["menu", lambda x: x]]
mode: "end"
```



**MDBottomAppBar:**

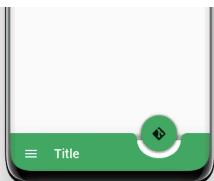
```
MDToolbar:
 title: "Title"
 icon: "git"
 type: "bottom"
 left_action_items: [["menu", lambda x: x]]
 mode: "free-end"
```



**Custom color**

```
MDBottomAppBar:
 md_bg_color: 0, 1, 0, 1

MDToolbar:
 title: "Title"
 icon: "git"
 type: "bottom"
 left_action_items: [["menu", lambda x: x]]
 icon_color: 0, 1, 0, 1
```



## MDToolbar with Menus

A Toolbar without Menus is not particularly useful. However, the `MDDropdownMenu` works well with the standard `MDToolbar` to provide this functionality, as shown in the image below.

### See also:

See the [MDDropdownMenu documentation](#) for details of how to implement this.

## Tooltips

You can add MDTooltips to the Toolbar icons by adding a text string to the toolbar item, as shown below

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.snackbar import Snackbar

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "MDToolbar"
 left_action_items: [["menu", "This is the navigation"]]
 right_action_items: [["dots-vertical", lambda x: app.callback(x), "this is the\u202aMore Actions"]]

 MDLabel:
 text: "Content"
 halign: "center"
 ...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def callback(self, button):
 Snackbar(text="Hello World").open()

Test().run()
```

### See also:

[Components-Bottom-App-Bar](#)

## API - kivymd.uix.toolbar

```
class kivymd.uix.toolbar.MDActionBottomAppBarButton(**kwargs)
 Base class for all round buttons, bringing in the appropriate on-touch behavior
```

```
class kivymd.uix.toolbar.MDActionTopAppBarButton(**kwargs)
 Base class for all round buttons, bringing in the appropriate on-touch behavior
```

```
class kivymd.uix.toolbar.NotchedBox(**kw)
```

*FakeRectangularElevationBehavior` is a shadow mockup for widgets. Improves performance using cached images inside `kivymd.images dir*

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
 ThemableBehavior,
 FakeCircularElevationBehavior,
 SpecificBackgroundColorBehavior,
 # here you add the other front end classes for the widget front_end,
):
 [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

*FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
 MDCard,
 FakeRectangularElevationBehavior
):
 [...]
```

---

**Note:** About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

---

### elevation

Elevation value.

*elevation* is an [NumericProperty](#) and defaults to 6.

### notch\_radius

### notch\_center\_x

```
class kivymd.uix.toolbar.MDToolbar(**kwargs)
```

## Events

**on\_action\_button** Method for the button used for the `MDBottomAppBar` class.

#### **left\_action\_items**

The icons on the left of the toolbar. To add one, append a list like the following:

```
left_action_items: [`'icon_name'`, callback, tooltip text]
```

where `'icon_name'` is a string that corresponds to an icon definition, `callback` is the function called on a touch release event and `tooltip text` is the text to be displayed in the tooltip. Both the `callback` and `tooltip text` are optional but the order must be preserved.

`left_action_items` is an `ListProperty` and defaults to `[]`.

#### **right\_action\_items**

The icons on the right of the toolbar. Works the same way as `left_action_items`.

`right_action_items` is an `ListProperty` and defaults to `[]`.

#### **title**

Text toolbar.

`title` is an `StringProperty` and defaults to ''.

#### **anchor\_title**

Position toolbar title. Available options are: `'left'`, `'center'`, `'right'`.

`anchor_title` is an `OptionProperty` and defaults to `'left'`.

#### **mode**

Floating button position. Only for `MDBottomAppBar` class. Available options are: `'free-end'`, `'free-center'`, `'end'`, `'center'`.

`mode` is an `OptionProperty` and defaults to `'center'`.

#### **round**

Rounding the corners at the notch for a button. Only for `MDBottomAppBar` class.

`round` is an `NumericProperty` and defaults to `'10dp'`.

#### **icon**

Floating button. Only for `MDBottomAppBar` class.

`icon` is an `StringProperty` and defaults to `'android'`.

#### **icon\_color**

Color action button. Only for `MDBottomAppBar` class.

`icon_color` is an `ColorProperty` and defaults to `[]`.

#### **type**

When using the `MDBottomAppBar` class, the parameter `type` must be set to `'bottom'`:

```
MDBottomAppBar:
```

```
 MDToolbar:
 type: "bottom"
```

Available options are: `'top'`, `'bottom'`.

`type` is an `OptionProperty` and defaults to `'top'`.

#### **opposite\_colors**

`on_type(self, instance, value)`

```
on_action_button(self, *args)
on_md_bg_color(self, instance, value)
on_left_action_items(self, instance, value)
on_right_action_items(self, instance, value)
set_md_bg_color(self, instance, value)
update_action_bar(self, action_bar, action_bar_items)
update_md_bg_color(self, *args)
update_opposite_colors(self, instance, value)
update_action_bar_text_colors(self, *args)
on_icon(self, instance, value)
on_icon_color(self, instance, value)
on_mode(self, instance, value)
remove_notch(self)
set_notch(self)
remove_shadow(self)
set_shadow(self, *args)
```

**class kivymd.uix.toolbar.MDBottomAppBar(\*\*kwargs)**

Float layout class. See module documentation for more information.

#### md\_bg\_color

Color toolbar.

*md\_bg\_color* is an [ColorProperty](#) and defaults to [0, 0, 0, 0].

#### add\_widget(self, widget, index=0, canvas=None)

Add a new widget as a child of this widget.

##### Parameters

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

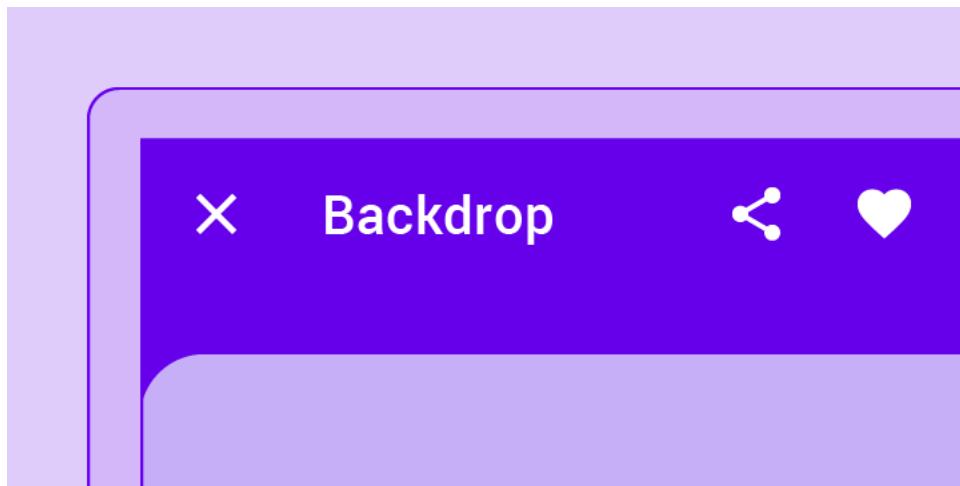
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

### 2.3.7 Backdrop

See also:

Material Design spec, Backdrop

Skeleton layout for using `MDBackdrop`:



#### Usage

```
<Root>:
 MDBackdrop:
 MDBackdropBackLayer:
 ContentForBackdropBackLayer:
 MDBackdropFrontLayer:
 ContentForBackdropFrontLayer:
```

#### Example

```
from kivy.lang import Builder
from kivy.uix.screenmanager import Screen

from kivymd.app import MDApp

Your layouts.
Builder.load_string(
 """
```

(continues on next page)

(continued from previous page)

```
#:import Window kivy.core.window.Window
#:import IconLeftWidget kivymd.uix.list.IconLeftWidget

<ItemBackdropFrontLayer@TwoLineAvatarListItem>
 icon: "android"

 IconLeftWidget:
 icon: root.icon

<MyBackdropFrontLayer@ItemBackdropFrontLayer>
 backdrop: None
 text: "Lower the front layer"
 secondary_text: " by 50 %"
 icon: "transfer-down"
 on_press: root.backdrop.open(-Window.height / 2)
 pos_hint: {"top": 1}
 _no_ripple_effect: True

<MyBackdropBackLayer@Image>
 size_hint: .8, .8
 source: "data/logo/kivy-icon-512.png"
 pos_hint: {"center_x": .5, "center_y": .6}
...
)

Usage example of MDBackdrop.
Builder.load_string(
 """
<ExampleBackdrop>

 MDBackdrop:
 id: backdrop
 left_action_items: [['menu', lambda x: self.open()]]
 title: "Example Backdrop"
 radius_left: "25dp"
 radius_right: "0dp"
 header_text: "Menu:"

 MDBackdropBackLayer:
 MyBackdropBackLayer:
 id: backlayer

 MDBackdropFrontLayer:
 MyBackdropFrontLayer:
 backdrop: backdrop
...
)
```

(continues on next page)

(continued from previous page)

```

class ExampleBackdrop(Screen):
 pass

class TestBackdrop(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)

 def build(self):
 return ExampleBackdrop()

TestBackdrop().run()

```

---

**Note:** See full example

---

## API - kivymd.uix.backdrop

```
class kivymd.uix.backdrop.MDBackdrop(**kwargs)
```

### Events

**on\_open** When the front layer drops.

**on\_close** When the front layer rises.

### padding

Padding for contents of the front layer.

*padding* is an `ListProperty` and defaults to `[0, 0, 0, 0]`.

### left\_action\_items

The icons and methods left of the `kivymd.uix.toolbar.MDToolbar` in back layer. For more information, see the `kivymd.uix.toolbar.MDToolbar` module and `left_action_items` parameter.

*left\_action\_items* is an `ListProperty` and defaults to `[]`.

### right\_action\_items

Works the same way as `left_action_items`.

*right\_action\_items* is an `ListProperty` and defaults to `[]`.

### title

See the `kivymd.uix.toolbar.MDToolbar.title` parameter.

*title* is an `StringProperty` and defaults to ''.

### back\_layer\_color

Background color of back layer.

*back\_layer\_color* is an `ColorProperty` and defaults to `None`.

**front\_layer\_color**

Background color of front layer.

`front_layer_color` is an [ColorProperty](#) and defaults to *None*.

**radius\_left**

The value of the rounding radius of the upper left corner of the front layer.

`radius_left` is an [NumericProperty](#) and defaults to *16dp*.

**radius\_right**

The value of the rounding radius of the upper right corner of the front layer.

`radius_right` is an [NumericProperty](#) and defaults to *16dp*.

**header**

Whether to use a header above the contents of the front layer.

`header` is an [BooleanProperty](#) and defaults to *True*.

**header\_text**

Text of header.

`header_text` is an [StringProperty](#) and defaults to ‘Header’.

**close\_icon**

The name of the icon that will be installed on the toolbar on the left when opening the front layer.

`close_icon` is an [StringProperty](#) and defaults to ‘close’.

**on\_open(self)**

When the front layer drops.

**on\_close(self)**

When the front layer rises.

**on\_left\_action\_items(self, instance, value)**

**on\_header(self, instance, value)**

**open(self, open\_up\_to=0)**

Opens the front layer.

**Open\_up\_to** the height to which the front screen will be lowered; if equal to zero - falls to the bottom of the screen;

**close(self)**

Opens the front layer.

**animtion\_icon\_menu(self)**

**animtion\_icon\_close(self, instance\_animation, instance\_icon\_menu)**

**add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

### Parameters

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

```
class kivymd.uix.backdrop.MDBackdropToolbar(**kwargs)
```

#### Events

**on\_action\_button** Method for the button used for the MDBottomAppBar class.

```
class kivymd.uix.backdrop.MDBackdropFrontLayer(**kwargs)
```

Box layout class. See module documentation for more information.

```
class kivymd.uix.backdrop.MDBackdropBackLayer(**kwargs)
```

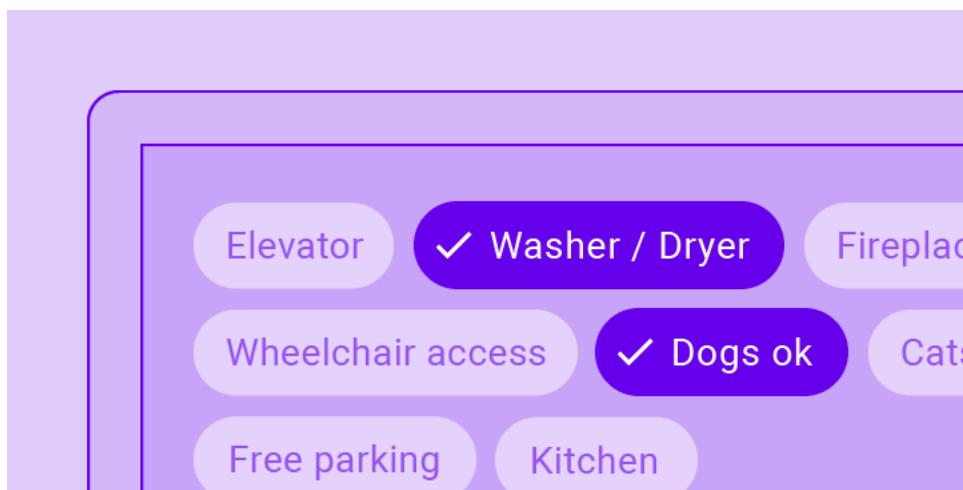
Box layout class. See module documentation for more information.

## 2.3.8 Chip

#### See also:

Material Design spec, Chips

**Chips are compact elements that represent an input, attribute, or action.**

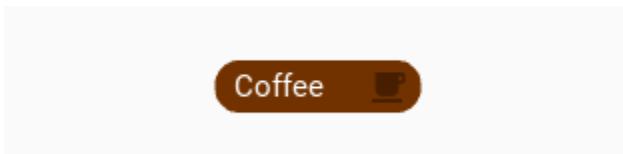


## Usage

```
MDChip:
 text: 'Coffee'
 color: .4470588235118, .1960787254902, 0, 1
 icon: 'coffee'
 on_release: app.callback_for_menu_items(self)
```

The user function takes two arguments - the object and the text of the chip:

```
def callback_for_menu_items(self, instance):
 print(instance)
```



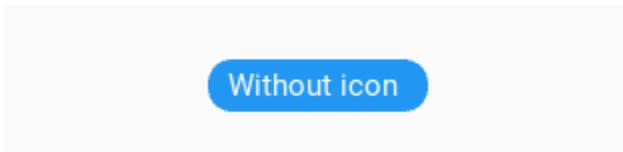
## Use custom icon

```
MDChip:
 text: 'Kivy'
 icon: 'data/logo/kivy-icon-256.png'
```



## Use without icon

```
MDChip:
 text: 'Without icon'
 icon: ''
```



## Chips with check

```
MDChip:
 text: 'Check with icon'
 icon: 'city'
 check: True
```

## Choose chip

```
MDChooseChip:

MDChip:
 text: 'Earth'
 icon: 'earth'
 selected_chip_color: .21176470535294, .098039627451, 1, 1

MDChip:
 text: 'Face'
 icon: 'face'
 selected_chip_color: .21176470535294, .098039627451, 1, 1

MDChip:
 text: 'Facebook'
 icon: 'facebook'
 selected_chip_color: .21176470535294, .098039627451, 1, 1
```

---

**Note:** See full example

---

## API - kivymd.uix.chip

**class kivymd.uix.chip.MDChip(\*\*kwargs)**

This `mixin` class provides `Button` behavior. Please see the `button behaviors` module documentation for more information.

### Events

`on_press` Fired when the button is pressed.

`on_release` Fired when the button is released (i.e. the touch/click that pressed the button goes away).

### text

Chip text.

`text` is an `StringProperty` and defaults to ''.

### icon

Chip icon.

`icon` is an `StringProperty` and defaults to ‘checkbox-blank-circle’.

**color**

Chip color in `rgba` format.

`color` is an `ColorProperty` and defaults to `None`.

**text\_color**

Chip’s text color in `rgba` format.

`text_color` is an `ColorProperty` and defaults to `None`.

**icon\_color**

Chip’s icon color in `rgba` format.

`icon_color` is an `ColorProperty` and defaults to `None`.

**check**

If True, a checkmark is added to the left when touch to the chip.

`check` is an `BooleanProperty` and defaults to `False`.

**radius**

Corner radius values.

`radius` is an `ListProperty` and defaults to ‘[dp(12),]’.

**selected\_chip\_color**

The color of the chip that is currently selected in `rgba` format.

`selected_chip_color` is an `ColorProperty` and defaults to `None`.

**set\_color(self, interval)**

**on\_icon(self, instance, value)**

**on\_touch\_down(self, touch)**

Receive a touch down event.

**Parameters**

**touch: MotionEvent class** Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

**Returns** bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

**class kivymd.uix.chip.MDChooseChip(\*\*kwargs)**

Stack layout class. See module documentation for more information.

**add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

**Parameters**

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget’s canvas to. Can be ‘before’, ‘after’ or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

### 2.3.9 Grid Layout

GridLayout class equivalent. Simplifies working with some widget properties. For example:

#### GridLayout

```
GridLayout:
 size_hint_y: None
 height: self.minimum_height

 canvas:
 Color:
 rgba: app.theme_cls.primary_color
 Rectangle:
 pos: self.pos
 size: self.size
```

#### MDGridLayout

```
MDGridLayout:
 adaptive_height: True
 md_bg_color: app.theme_cls.primary_color
```

**Available options are:**

- *adaptive\_height*
- *adaptive\_width*
- *adaptive\_size*

### adaptive\_height

```
adaptive_height: True
```

Equivalent

```
size_hint_y: None
height: self.minimum_height
```

### adaptive\_width

```
adaptive_width: True
```

Equivalent

```
size_hint_x: None
width: self.minimum_width
```

### adaptive\_size

```
adaptive_size: True
```

Equivalent

```
size_hint: None, None
size: self.minimum_size
```

## API - kivymd.uix.gridlayout

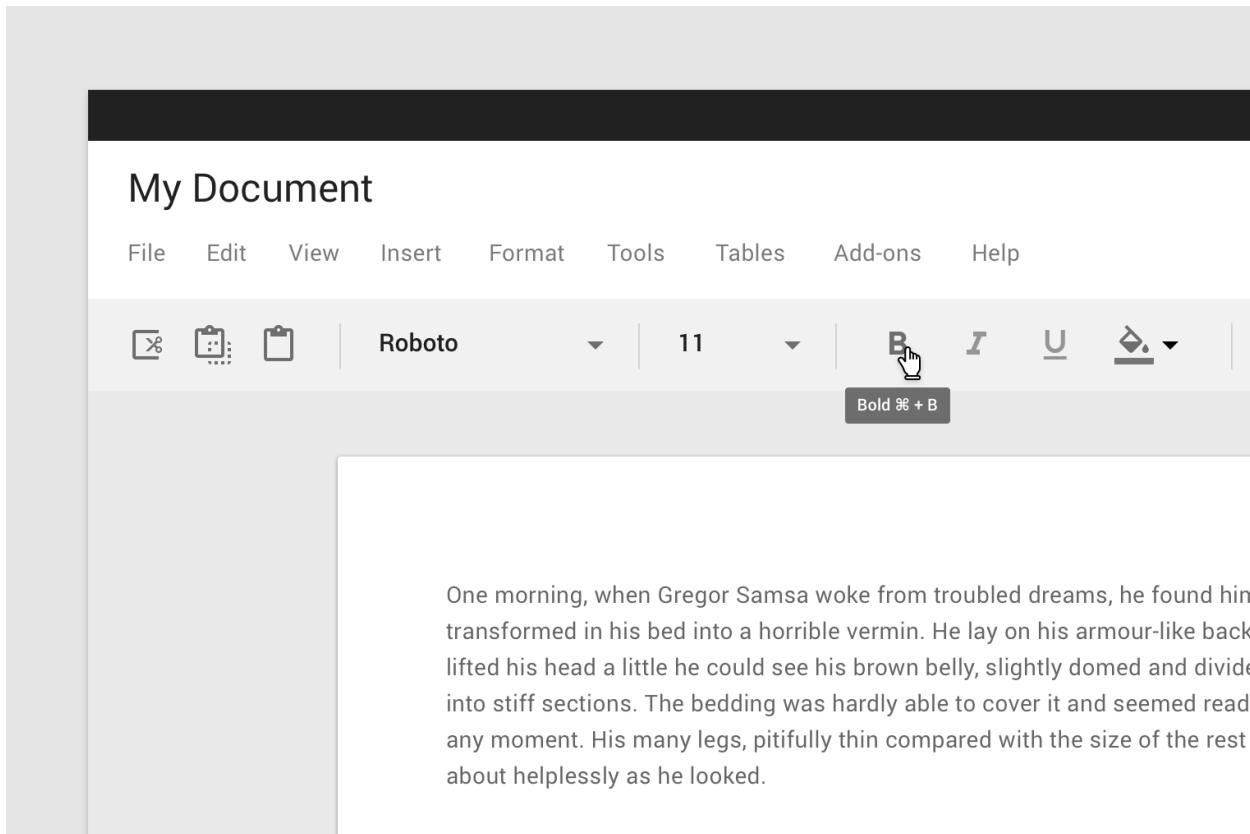
```
class kivymd.uix.gridlayout.MDGridLayout(**kwargs)
 Grid layout class. See module documentation for more information.
```

### 2.3.10 Tooltip

#### See also:

Material Design spec, Tooltips

**Tooltips display informative text when users hover over, focus on, or tap an element.**



To use the `MDTooltip` class, you must create a new class inherited from the `MDTooltip` class:

In Kv-language:

```
<TooltipMDIconButton@MDIconButton+MDTooltip>
```

In Python code:

```
class TooltipMDIconButton(MDIconButton, MDTooltip):
 pass
```

**Warning:** `MDTooltip` only works correctly with button and label classes.

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<TooltipMDIconButton@MDIconButton+MDTooltip>
```

Screen:

(continues on next page)

(continued from previous page)

```
TooltipMDIconButton:
 icon: "language-python"
 tooltip_text: self.icon
 pos_hint: {"center_x": .5, "center_y": .5}
...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```

---

**Note:** The behavior of tooltips on desktop and mobile devices is different. For more detailed information, [click here](#).

---

## API - kivymd.uix.tooltip

```
class kivymd.uix.tooltip.MDTooltip(**kwargs)
```

### Events

**on\_enter** Called when mouse enters the bbox of the widget AND the widget is visible

**on\_leave** Called when the mouse exits the widget AND the widget is visible

#### tooltip\_bg\_color

Tooltip background color in rgba format.

*tooltip\_bg\_color* is an `ColorProperty` and defaults to `None`.

#### tooltip\_text\_color

Tooltip text color in rgba format.

*tooltip\_text\_color* is an `ColorProperty` and defaults to `None`.

#### tooltip\_text

Tooltip text.

*tooltip\_text* is an `StringProperty` and defaults to ''.

#### tooltip\_font\_style

Tooltip font style. Available options are: 'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'Subtitle1', 'Subtitle2', 'Body1', 'Body2', 'Button', 'Caption', 'Overline', 'Icon'.

*tooltip\_font\_style* is an `OptionProperty` and defaults to 'Caption'.

#### tooltip\_radius

Corner radius values.

*radius* is an `ListProperty` and defaults to [dp(7),].

```
tooltip_display_delay
 Tooltip display delay.

 tooltip_display_delay is an BoundedNumericProperty and defaults to 0, min of 0 & max of 4. This
 property only works on desktop.

shift_y
 Y-offset of tooltip text.

 shift_y is an StringProperty and defaults to 0.

delete_clock(self, widget, touch, *args)
adjust_tooltip_position(self, x, y)
 Returns the coordinates of the tooltip that fit into the borders of the screen.

display_tooltip(self, interval)
animation_tooltip_show(self, interval)
remove_tooltip(self, *args)
on_long_touch(self, touch, *args)
 Called when the widget is pressed for a long time.

on_enter(self, *args)
 See on_enter method in HoverBehavior class.

on_leave(self)
 See on_leave method in HoverBehavior class.

on_show(self)

class kivymd.uix.tooltip.MDTooltipViewClass(**kwargs)
 Box layout class. See module documentation for more information.

tooltip_bg_color
 See tooltip_bg_color.

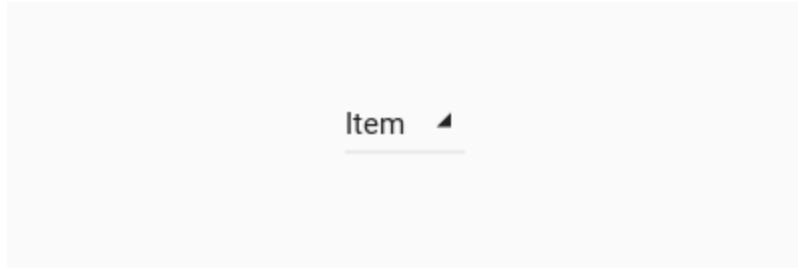
tooltip_text_color
 See tooltip_text_color.

tooltip_text
 See tooltip_text.

tooltip_font_style
 See tooltip_font_style.

tooltip_radius
 See tooltip_radius.
```

### 2.3.11 Dropdown Item



## Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
Screen

 MDDropDownItem:
 id: drop_item
 pos_hint: {'center_x': .5, 'center_y': .5}
 text: 'Item'
 on_release: self.set_item("New Item")
'''

class Test(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)

 def build(self):
 return self.screen

Test().run()
```

### See also:

Work with the class `MDDropdownMenu` see here

### API - `kivymd.uix.dropdownitem`

```
class kivymd.uix.dropdownitem.MDDropDownItem(**kwargs)
FakeRectangularElevationBehavior is a shadow mockup for widgets. Improves performance using cached images inside `kivymd.images` dir
```

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
 ThemableBehavior,
 FakeCircularElevationBehavior,
 SpecificBackgroundColorBehavior,
 # here you add the other front end classes for the widget front_end,
):
 [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

*FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
 MDCard,
 FakeRectangularElevationBehavior
):
 [...]
```

---

**Note:** About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

---

#### **text**

Text item.

*text* is a **StringProperty** and defaults to “”.

#### **current\_item**

Current name item.

*current\_item* is a **StringProperty** and defaults to “”.

#### **font\_size**

Item font size.

*font\_size* is a **NumericProperty** and defaults to '16sp'.

#### **on\_text(self, instance, value)**

#### **set\_item(self, name\_item)**

Sets new text for an item.

### 2.3.12 Float Layout

FloatLayout class equivalent. Simplifies working with some widget properties. For example:

## FloatLayout

```
FloatLayout:
 canvas:
 Color:
 rgba: app.theme_cls.primary_color
 RoundedRectangle:
 pos: self.pos
 size: self.size
 radius: [25, 0, 0, 0]
```

## MDFloatLayout

```
MDFloatLayout:
 radius: [25, 0, 0, 0]
 md_bg_color: app.theme_cls.primary_color
```

**Warning:** For a `FloatLayout`, the `minimum_size` attributes are always 0, so you cannot use `adaptive_size` and related options.

## API - `kivymd.uix.floatlayout`

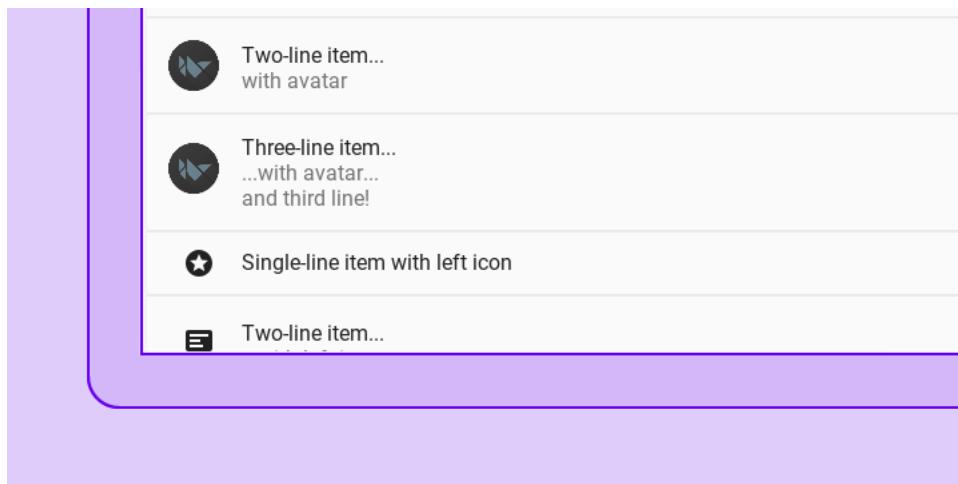
```
class kivymd.uix.floatlayout.MDFloatLayout(**kwargs)
 Float layout class. See module documentation for more information.
```

### 2.3.13 List

#### See also:

Material Design spec, Lists

**Lists are continuous, vertical indexes of text or images.**



The class `MDList` in combination with a `BaseListItem` like `OneLineListItem` will create a list that expands as items are added to it, working nicely with *Kivy's ScrollView*.

Due to the variety in sizes and controls in the *Material Design spec*, this module suffers from a certain level of complexity to keep the widgets compliant, flexible and performant.

For this *KivyMD* provides list items that try to cover the most common usecases, when those are insufficient, there's a base class called `BaseListItem` which you can use to create your own list items. This documentation will only cover the provided ones, for custom implementations please refer to this module's source code.

*KivyMD* provides the following list items classes for use:

### Text only ListItems

- `OneLineListItem`
- `TwoLineListItem`
- `ThreeLineListItem`

### ListItems with widget containers

These widgets will take other widgets that inherit from `ILeftBody`, `ILeftBodyTouch`, `IRightBody` or `IRightBodyTouch` and put them in their corresponding container.

As the name implies, `ILeftBody` and `IRightBody` will signal that the widget goes into the left or right container, respectively.

`ILeftBodyTouch` and `IRightBodyTouch` do the same thing, except these widgets will also receive touch events that occur within their surfaces.

*KivyMD* provides base classes such as `ImageLeftWidget`, `ImageRightWidget`, `IconRightWidget`, `IconLeftWidget`, based on the above classes.

Allows the use of items with custom widgets on the left.

- *OneLineAvatarListItem*
- *TwoLineAvatarListItem*
- *ThreeLineAvatarListItem*
- *OneLineIconListItem*
- *TwoLineIconListItem*
- *ThreeLineIconListItem*

It allows the use of elements with custom widgets on the left and the right.

- *OneLineAvatarIconListItem*
- *TwoLineAvatarIconListItem*
- *ThreeLineAvatarIconListItem*

## Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.list import OneLineListItem

KV = '''
ScrollView:

 MDList:
 id: container
'''

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for i in range(20):
 self.root.ids.container.add_widget(
 OneLineListItem(text=f"Single-line item {i}")
)

Test().run()
```

## Events of List

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = """
ScrollView:

 MDList:

 OneLineAvatarIconListItem:
 on_release: print("Click! ")

 IconLeftWidget:
 icon: "github"

 OneLineAvatarIconListItem:
 on_release: print("Click 2!")

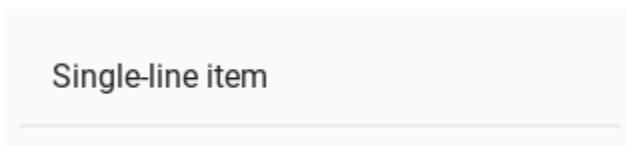
 IconLeftWidget:
 icon: "gitlab"
 ...

class MainApp(MDApp):
 def build(self):
 return Builder.load_string(KV)

MainApp().run()
```

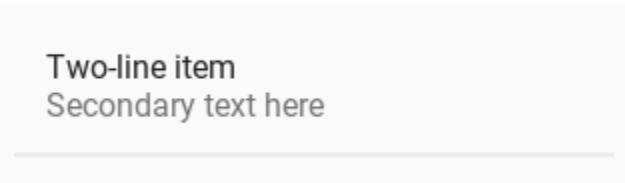
## OneLineListItem

```
OneLineListItem:
 text: "Single-line item"
```



## TwoLineListItem

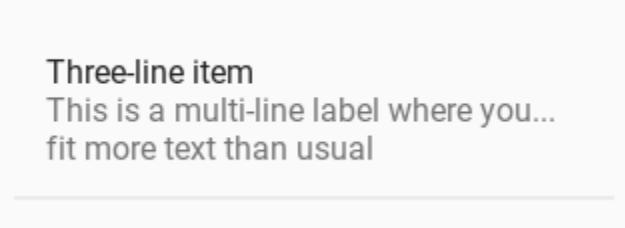
```
TwoLineListItem:
 text: "Two-line item"
 secondary_text: "Secondary text here"
```



Two-line item  
Secondary text here

## ThreeLineListItem

```
ThreeLineListItem:
 text: "Three-line item"
 secondary_text: "This is a multi-line label where you can"
 tertiary_text: "fit more text than usual"
```



Three-line item  
This is a multi-line label where you...  
fit more text than usual

## OneLineAvatarListItem

```
OneLineAvatarListItem:
 text: "Single-line item with avatar"

ImageLeftWidget:
 source: "data/logo/kivy-icon-256.png"
```



### TwoLineAvatarListItem

```
TwoLineAvatarListItem:
 text: "Two-line item with avatar"
 secondary_text: "Secondary text here"

ImageLeftWidget:
 source: "data/logo/kivy-icon-256.png"
```



Two-line item with avatar  
Secondary text here

### ThreeLineAvatarListItem

```
ThreeLineAvatarListItem:
 text: "Three-line item with avatar"
 secondary_text: "Secondary text here"
 tertiary_text: "fit more text than usual"

ImageLeftWidget:
 source: "data/logo/kivy-icon-256.png"
```



Three-line item with avatar  
Secondary text here  
fit more text than usual

### OneLineIconListItem

```
OneLineIconListItem:
 text: "Single-line item with avatar"

IconLeftWidget:
 icon: "language-python"
```



Single-line item with avatar

### TwoLineIconListItem

```
TwoLineIconListItem:
 text: "Two-line item with avatar"
 secondary_text: "Secondary text here"

 IconLeftWidget:
 icon: "language-python"
```



Two-line item with avatar  
Secondary text here

### ThreeLineIconListItem

```
ThreeLineIconListItem:
 text: "Three-line item with avatar"
 secondary_text: "Secondary text here"
 tertiary_text: "fit more text than usual"

 IconLeftWidget:
 icon: "language-python"
```



Three-line item with avatar  
Secondary text here  
fit more text than usual

### OneLineAvatarIconListItem

```
OneLineAvatarIconListItem:
 text: "One-line item with avatar"

 IconLeftWidget:
 icon: "plus"

 IconRightWidget:
 icon: "minus"
```

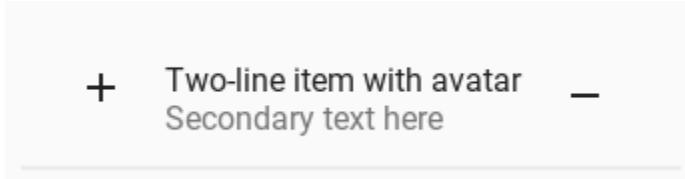
+ One-line item with avatar -

## TwoLineAvatarIconListItem

```
TwoLineAvatarIconListItem:
 text: "Two-line item with avatar"
 secondary_text: "Secondary text here"

IconLeftWidget:
 icon: "plus"

IconRightWidget:
 icon: "minus"
```



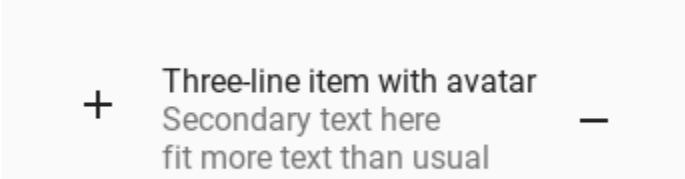
+ Two-line item with avatar  
Secondary text here -

## ThreeLineAvatarIconListItem

```
ThreeLineAvatarIconListItem:
 text: "Three-line item with avatar"
 secondary_text: "Secondary text here"
 tertiary_text: "fit more text than usual"

IconLeftWidget:
 icon: "plus"

IconRightWidget:
 icon: "minus"
```



+ Three-line item with avatar  
Secondary text here  
fit more text than usual -

## Custom list item

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.list import IRightBodyTouch, OneLineAvatarIconListItem
from kivymd.uix.selectioncontrol import MDCheckbox
from kivymd.icon_definitions import md_icons
```

(continues on next page)

(continued from previous page)

```
KV = """
<ListItemWithCheckbox>:

 IconLeftWidget:
 icon: root.icon

 RightCheckbox:

BoxLayout:

 ScrollView:

 MDList:
 id: scroll
 ...

class ListItemWithCheckbox(OneLineAvatarIconListItem):
 "Custom list item."

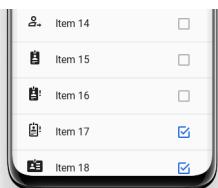
 icon = StringProperty("android")

class RightCheckbox(IRightBodyTouch, MDCheckbox):
 "Custom right container."

class MainApp(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 icons = list(md_icons.keys())
 for i in range(30):
 self.root.ids.scroll.add_widget(
 ListItemWithCheckbox(text=f"Item {i}", icon=icons[i])
)

MainApp().run()
```



```
from kivy.lang import Builder
```

(continues on next page)

(continued from previous page)

```

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBBoxLayout
from kivymd.uix.list import IRightBodyTouch

KV = """
OneLineAvatarIconListItem:
 text: "One-line item with avatar"
 on_size:
 self.ids._right_container.width = container.width
 self.ids._right_container.x = container.width

 IconLeftWidget:
 icon: "cog"

 Container:
 id: container

 MDIconButton:
 icon: "minus"

 MDIconButton:
 icon: "plus"
 ...

class Container(IRightBodyTouch, MDBBoxLayout):
 adaptive_width = True

class MainApp(MDApp):
 def build(self):
 return Builder.load_string(KV)

MainApp().run()

```



## Behavior

When using the *AvatarListItem* and *IconListItem* classes, when an icon is clicked, the event of this icon is triggered:

```

OneLineIconListItem:
 text: "Single-line item with icon"

 IconLeftWidget:
 icon: "language-python"

```

You can disable the icon event using the *WithoutTouch* classes:

```
OneLineIconListItem:
 text: "Single-line item with icon"

IconLeftWidgetWithoutTouch:
 icon: "language-python"
```

## API - kivymd.uix.list

**class kivymd.uix.list.MDList(\*\*kwargs)**

ListItem container. Best used in conjunction with a `kivy.uixScrollView`.

When adding (or removing) a widget, it will resize itself to fit its children, plus top and bottom paddings as described by the *MD* spec.

**add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

### Parameters

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

**remove\_widget(self, widget)**

Remove a widget from the children of this widget.

### Parameters

**widget: Widget** Widget to remove from our children list.

```
>>> from kivy.uix.button import Button
>>> root = Widget()
>>> button = Button()
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

**class kivymd.uix.list.ILeftBodyTouch**

Same as ILeftBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect.

**class kivymd.uix.list.IRightBodyTouch**

Same as IRightBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect

**class kivymd.uix.list.OneLineListItem(\*\*kwargs)**

A one line list item.

**class kivymd.uix.list.TwoLineListItem(\*\*kwargs)**

A two line list item.

**class kivymd.uix.list.ThreeLineListItem(\*\*kwargs)**

A three line list item.

**class kivymd.uix.list.OneLineAvatarListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.TwoLineAvatarListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.ThreeLineAvatarListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.OneLineIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.TwoLineIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.ThreeLineIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.OneLineRightIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.TwoLineRightIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.ThreeLineRightIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.OneLineAvatarIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

**class kivymd.uix.list.TwoLineAvatarIconListItem(\*\*kwargs)**

Overrides add\_widget in a ListItem to include support for I\*Body widgets when the appropriate containers are present.

```
class kivymd.uix.list.ThreeLineAvatarIconListItem(**kwargs)
 Overrides add_widget in a ListItem to include support for I*Body widgets when the appropriate containers are present.

class kivymd.uix.list.ImageLeftWidget(**kwargs)
 Class implements a circular ripple effect.

class kivymd.uix.list.ImageRightWidget(**kwargs)
 Class implements a circular ripple effect.

class kivymd.uix.list.IconRightWidget(**kwargs)
 Same as IRightBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect

class kivymd.uix.list.IconLeftWidget(**kwargs)
 Same as ILeftBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect.

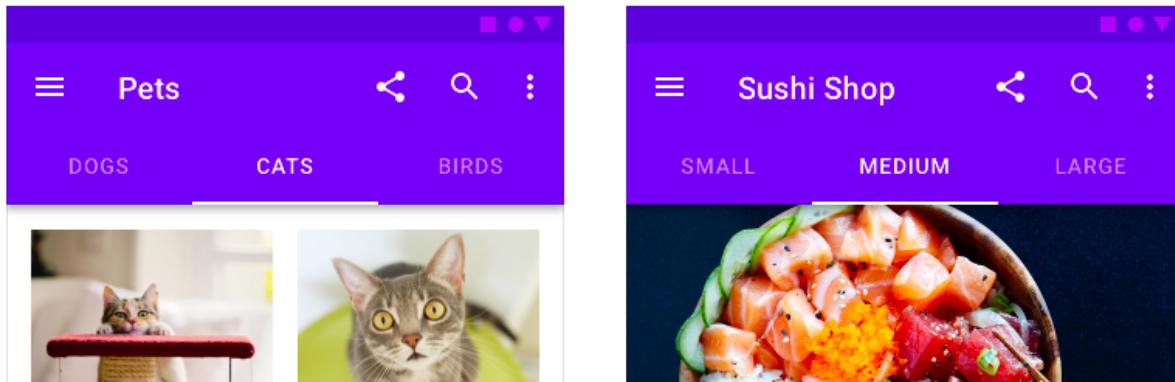
class kivymd.uix.list.CheckboxLeftWidget(**kwargs)
 Same as ILeftBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect.
```

## 2.3.14 Tabs

See also:

Material Design spec, Tabs

**Tabs organize content across different screens, data sets, and other interactions.**



---

**Note:** Module provides tabs in the form of icons or text.

---

## Usage

To create a tab, you must create a new class that inherits from the `MDTabsBase` class and the *Kivy* container, in which you will create content for the tab.

```
class Tab(MDFloatLayout, MDTabsBase):
 "Class implementing content for a tab."
 content_text = StringProperty("")
```

```
<Tab>
 content_text
 MDLabel:
 text: root.content_text
 pos_hint: {"center_x": .5, "center_y": .5}
```

All tabs must be contained inside a `MDTabs` widget:

```
Root:
 MDTabs:
 Tab:
 title: "Tab 1"
 content_text: f"This is an example text for {self.title}"
 Tab:
 title: "Tab 2"
 content_text: f"This is an example text for {self.title}"
 ...

```

## Example with tab icon

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsBase
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.icon_definitions import md_icons

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs
 on_tab_switch: app.on_tab_switch(*args)
```

(continues on next page)

(continued from previous page)

```
<Tab>

MDIconButton:
 id: icon
 icon: root.icon
 user_font_size: "48sp"
 pos_hint: {"center_x": .5, "center_y": .5}
 ...

class Tab(MDFloatLayout, MDTabsBase):
 "Class implementing content for a tab."

class Example(MDApp):
 icons = list(md_icons.keys())[15:30]

 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for tab_name in self.icons:
 self.root.ids.tabs.add_widget(Tab(icon=tab_name))

 def on_tab_switch(
 self, instance_tabs, instance_tab, instance_tab_label, tab_text
):
 """
 Called when switching tabs.

 :type instance_tabs: <kivymd.uix.tab.MDTabs object>;
 :param instance_tab: <__main__.Tab object>;
 :param instance_tab_label: <kivymd.uix.tab.MDTabsLabel object>;
 :param tab_text: text or name icon of tab;
 """

 # get the tab icon.
 count_icon = instance_tab.icon
 # print it on shell/bash.
 print(f"Welcome to {count_icon}' tab'")

Example().run()
```

## Example with tab text

**Note:** The `MDTabsBase` class has an `icon` parameter and, by default, tries to find the name of the icon in the file `kivymd/icon_definitions.py`.

If the name of the icon is not found, the class will send a message stating that the icon could not be found.

If the tab has no icon, title or `tab_label_text`, the class will raise a `ValueError`.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.tab import MDTabsBase

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs
 on_tab_switch: app.on_tab_switch(*args)

<Tab>

 MDLabel:
 id: label
 text: "Tab 0"
 halign: "center"
 ...

class Tab(MDFloatLayout, MDTabsBase):
 "Class implementing content for a tab."

class Example(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for i in range(20):
 self.root.ids.tabs.add_widget(Tab(title=f"Tab {i}"))

 def on_tab_switch(
 self, instance_tabs, instance_tab, instance_tab_label, tab_text
):
 "Called when switching tabs."
```

(continues on next page)

(continued from previous page)

```
:type instance_tabs: <kivymd.uix.tab.MDTabs object>;
:param instance_tab: <__main__.Tab object>;
:param instance_tab_label: <kivymd.uix.tab.MDTabsLabel object>;
:param tab_text: text or name icon of tab;
"""

instance_tab.ids.label.text = tab_text
```

Example().run()

### Example with tab icon and text

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsBase
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.icon_definitions import md_icons

KV = '''
MDBBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs
'''

class Tab(MDFloatLayout, MDTabsBase):
 pass

class Example(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for name_tab in list(md_icons.keys())[15:30]:
 self.root.ids.tabs.add_widget(Tab(icon=name_tab, title=name_tab))

Example().run()
```

## Example Tabs

account-cancel-outline    account-cash    account-cash-outline    account-check    account-check-outline

### Dynamic tab management

```
from kivy.lang import Builder
from kivy.uix.scrollview import ScrollView

from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsBase

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs

<Tab>

 MDList:

 MDBBoxLayout:
 adaptive_height: True

 MDFlatButton:
 text: "ADD TAB"
 on_release: app.add_tab()

 MDFlatButton:
 text: "REMOVE LAST TAB"
 on_release: app.remove_tab()

 MDFlatButton:
 text: "GET TAB LIST"
 on_release: app.get_tab_list()
 ...
'''

class Tab(ScrollView, MDTabsBase):
 """Class implementing content for a tab."""

class Example(MDApp):
```

(continues on next page)

(continued from previous page)

```

index = 0

def build(self):
 return Builder.load_string(KV)

def on_start(self):
 self.add_tab()

def get_tab_list(self):
 "Prints a list of tab objects."
 print(self.root.ids.tabs.get_tab_list())

def add_tab(self):
 self.index += 1
 self.root.ids.tabs.add_widget(Tab(text=f"{self.index} tab"))

def remove_tab(self):
 if self.index > 1:
 self.index -= 1
 self.root.ids.tabs.remove_widget(
 self.root.ids.tabs.get_tab_list()[-1]
)

```

Example().run()

### Use on\_ref\_press method

You can use markup for the text of the tabs and use the on\_ref\_press method accordingly:

```

from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.font_definitions import fonts
from kivymd.uix.tab import MDTabsBase
from kivymd.icon_definitions import md_icons

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs
 on_ref_press: app.on_ref_press(*args)

```

(continues on next page)

(continued from previous page)

```
<Tab>

MDIconButton:
 id: icon
 icon: app.icons[0]
 user_font_size: "48sp"
 pos_hint: {"center_x": .5, "center_y": .5}
 ...

class Tab(MDFloatLayout, MDTabsBase):
 "Class implementing content for a tab."

class Example(MDApp):
 icons = list(md_icons.keys())[15:30]

 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for name_tab in self.icons:
 self.root.ids.tabs.add_widget(
 Tab(
 text=f"[ref={name_tab}][font={fonts[-1]['fn_regular']}]{md_icons[
→'close']}[/font][/ref] {name_tab}"
)
)

 def on_ref_press(
 self,
 instance_tabs,
 instance_tab_label,
 instance_tab,
 instance_tab_bar,
 instance_carousel,
):
 """
 The method will be called when the ``on_ref_press`` event
 occurs when you, for example, use markup text for tabs.

 :param instance_tabs: <kivymd.uix.tab.MDTabs object>
 :param instance_tab_label: <kivymd.uix.tab.MDTabsLabel object>
 :param instance_tab: <__main__.Tab object>
 :param instance_tab_bar: <kivymd.uix.tab.MDTabsBar object>
 :param instance_carousel: <kivymd.uix.tab.MDTabsCarousel object>
 """

 # Removes a tab by clicking on the close icon on the left.
 for instance_tab in instance_carousel.slides:
```

(continues on next page)

(continued from previous page)

```
if instance_tab.text == instance_tab_label.text:
 instance_tabs.remove_widget(instance_tab_label)
 break
```

```
Example().run()
```

### Switching the tab by name

```
from kivy.lang import Builder
from kivymd.app import MDApp
from kivymd.icon_definitions import md_icons
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.tab import MDTabsBase

KV = '''
MDBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "Example Tabs"

 MDTabs:
 id: tabs

<Tab>
 MDBoxLayout:
 orientation: "vertical"
 pos_hint: {"center_x": .5, "center_y": .5}
 size_hint: None, None
 spacing: dp(48)
 MDIconButton:
 id: icon
 icon: "arrow-right"
 user_font_size: "48sp"

 on_release:
 app.switch_tab_by_name()

 MDIconButton:
 id: icon2
 icon: "page-next"
 user_font_size: "48sp"

 on_release:
 app.switch_tab_by_object()
'''
```

(continues on next page)

(continued from previous page)

```

class Tab(MDFloatLayout, MDTabsBase):
 "Class implementing content for a tab."

class Example(MDApp):
 icons = list(md_icons.keys())[15:30]

 def build(self):
 self.iter_list_names = iter(list(self.icons))
 root = Builder.load_string(KV)
 return root

 def on_start(self):
 for name_tab in list(self.icons):
 self.root.ids.tabs.add_widget(Tab(tab_label_text=name_tab))
 self.iter_list_objects = iter(list(self.root.ids.tabs.get_tab_list()))

 def switch_tab_by_object(self):
 try:
 x = next(self.iter_list_objects)
 print(f"Switch slide by object,
nex element to show: {[x]}")
 self.root.ids.tabs.switch_tab(x)
 except StopIteration:
 # reset the iterator an begin again.
 self.iter_list_objects = iter(list(self.root.ids.tabs.get_tab_list()))
 self.switch_tab_by_object()
 pass

 def switch_tab_by_name(self):
 "Switching the tab by name."

 try:
 x = next(self.iter_list_names)
 print(f"Switch slide by name,
nex element to show: {[x]}")
 self.root.ids.tabs.switch_tab(x)
 except StopIteration:
 # reset the iterator an begin again.
 self.iter_list_names = iter(list(self.icons))
 self.switch_tab_by_name()
 pass

Example().run()

```

## API - kivymd.uix.tab

**class** `kivymd.uix.tab.MDTabsBase(**kwargs)`

This class allow you to create a tab. You must create a new class that inherits from MDTabsBase. In this way you have total control over the views of your tabbed panel.

### **icon**

This property will set the Tab's Label Icon.

`icon` is an `StringProperty` and defaults to ''.

### **title\_icon\_mode**

This property sets the mode in which the tab's title and icon are shown.

`title_icon_mode` is an `OptionProperty` and defaults to '`Lead`'.

### **title**

This property will set the Name of the tab.

---

**Note:** As a side note.

All tabs have set `markup = True`. Thanks to this, you can use the kivy markup language to set a colorful and fully customizable tabs titles.

---

**Warning:** The material design requires that every title label is written in capital letters, because of this, the `string.upper()` will be applied to it's contents.

`title` is an `StringProperty` and defaults to ''.

### **title\_is\_capital**

This value controls whether if the title property should be converted to capital letters.

`title_is_capital` is an `BooleanProperty` and defaults to `True`.

### **text**

This property is the actual title of the tab. use the property `icon` and `title` to set this property correctly.

This property is kept public for specific and backward compatibility purposes.

`text` is an `StringProperty` and defaults to ''.

**Warning:** This property is deprecated, use `tab_label_text` instead.

### **tab\_label\_text**

This property is the actual title's Label of the tab. use the property `icon` and `title` to set this property correctly.

This property is kept public for specific and backward compatibility purposes.

`tab_label_text` is an `StringProperty` and defaults to ''.

### **tab\_label**

It is the label object reference of the tab.

`tab_label` is an `ObjectProperty` and defaults to `None`.

**tab\_label\_font\_style**

`tab_label_font_style` is an [AliasProperty](#) that behaves similar to an [OptionProperty](#).

This property's behavior allows the developer to use any new label style registered to the app.

This property will affect the Tab's Title Label widget.

`update_label_text(self, widget, value)`

`on_text(self, widget, text)`

**class kivymd.uix.tab.MDTabs(\*\*kwargs)**

You can use this class to create your own tabbed panel.

**Events**

`on_tab_switch` Called when switching tabs.

`on_slide_progress` Called while the slide is scrolling.

`on_ref_press` The method will be called when the `on_ref_press` event occurs when you, for example, use markup text for tabs.

**default\_tab**

Index of the default tab.

`default_tab` is an [NumericProperty](#) and defaults to `0`.

**tab\_bar\_height**

Height of the tab bar.

`tab_bar_height` is an [NumericProperty](#) and defaults to `'48dp'`.

**tab\_padding**

Padding of the tab bar.

`tab_padding` is an [ListProperty](#) and defaults to `[0, 0, 0, 0]`.

**tab\_indicator\_anim**

Tab indicator animation. If you want use animation set it to `True`.

`tab_indicator_anim` is an [BooleanProperty](#) and defaults to `False`.

**tab\_indicator\_height**

Height of the tab indicator.

`tab_indicator_height` is an [NumericProperty](#) and defaults to `'2dp'`.

**tab\_indicator\_type**

Type of tab indicator. Available options are: `'line'`, `'fill'`, `'round'`, `'line-rect'` and `'line-round'`.

`tab_indicator_type` is an [OptionProperty](#) and defaults to `'line'`.

**tab\_hint\_x**

This option affects the size of each child. if it's `True`, the size of each tab will be ignored and will use the size available by the container.

`tab_hint_x` is an [BooleanProperty](#) and defaults to `False`.

**anim\_duration**

Duration of the slide animation.

`anim_duration` is an [NumericProperty](#) and defaults to `0.2`.

**anim\_threshold**

Animation threshold allow you to change the tab indicator animation effect.

*anim\_threshold* is an `BoundedNumericProperty` and defaults to *0.8*.

**allow\_stretch**

If *True*, the tab will update dynamically (if `tab_hint_x` is *True*) to its content width, and wrap any text if the widget is wider than “*360dp*”.

If *False*, the tab won’t update to its maximum texture width. this means that the `fixed_tab_label_width` will be used as the label width. this will wrap any text inside to fit the fixed value.

*allow\_stretch* is an `BooleanProperty` and defaults to *True*.

**fixed\_tab\_label\_width**

If `allow_stretch` is *False*, the class will set this value as the width to all the tabs title label.

*fixed\_tab\_label\_width* is an `NumericProperty` and defaults to *140dp*.

**background\_color**

Background color of tabs in `rgba` format.

*background\_color* is an `ColorProperty` and defaults to *None*.

**underline\_color**

Underline color of tabs in `rgba` format.

*underline\_color* is an `ColorProperty` and defaults to *[0, 0, 0, 0]*.

**text\_color\_normal**

Text color of the label when it is not selected.

*text\_color\_normal* is an `ColorProperty` and defaults to *None*.

**text\_color\_active**

Text color of the label when it is selected.

*text\_color\_active* is an `ColorProperty` and defaults to *None*.

**elevation**

Tab value elevation.

**See also:**

Behaviors/Elevation

*elevation* is an `NumericProperty` and defaults to *0*.

**indicator\_color**

Color indicator in `rgba` format.

*indicator\_color* is an `ColorProperty` and defaults to *None*.

**lock\_swiping**

If *True* - disable switching tabs by swipe.

*lock\_swiping* is an `BooleanProperty` and defaults to *False*.

**font\_name**

Font name for tab text.

*font\_name* is an `StringProperty` and defaults to ‘*Roboto*’.

**ripple\_duration**

Ripple duration when long touching to tab.

*ripple\_duration* is an `NumericProperty` and defaults to *2*.

**no\_ripple\_effect**

Whether to use the ripple effect when tapping on a tab.

`no_ripple_effect` is an `BooleanProperty` and defaults to `True`.

**title\_icon\_mode**

This property sets the mode in which the tab's title and icon are shown.

`title_icon_mode` is an `OptionProperty` and defaults to '`Lead`'.

**force\_title\_icon\_mode**

If this property is set to `True`, it will force the class to update every tab inside the scroll view to the current `title_icon_mode`

`force_title_icon_mode` is an `BooleanProperty` and defaults to `True`.

**update\_icon\_color(self, instance, value)****switch\_tab(self, name\_tab, search\_by='text')**

This function switch between tabs `name_tab` can be either a String or a MDTabsBase.

`search_by` will look up through the properties of every tab.

If the value doesn't match, it will raise a `ValueError`.

**Search\_by options:** `text` : will search by the raw text of the label (`tab_label_text`) `icon` : will search by the `icon` property `title` : will search by the `title` property

**get\_tab\_list(self)**

Returns a list of tab objects.

**get\_slides(self)****add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

**Parameters**

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

**remove\_widget(self, widget)**

Remove a widget from the children of this widget.

**Parameters**

**widget: Widget** Widget to remove from our children list.

```
>>> from kivy.uix.button import Button
>>> root = Widget()
>>> button = Button()
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

**on\_slide\_progress(self, \*args)**

This event is deployed every available frame while the tab is scrolling.

**on\_carousel\_index(self, carousel, index)**

Called when the Tab index have changed.

This event is deployed by the built in carousel of the class.

**on\_ref\_press(self, \*args)**

This event will be launched every time the user press a markup enabled label with a link or reference inside.

**on\_tab\_switch(self, \*args)**

This event is launched every time the current tab is changed.

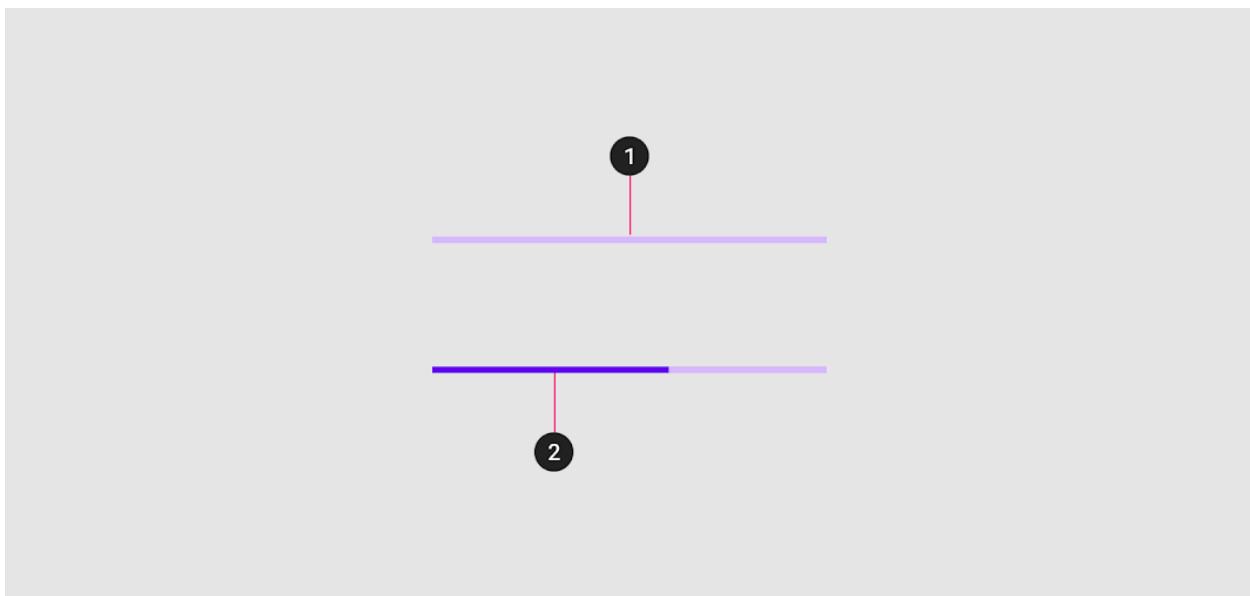
**on\_size(self, \*args)**

### 2.3.15 Progress Bar

**See also:**

Material Design spec, Progress indicators

**Progress indicators express an unspecified wait time or display the length of a process.**



KivyMD provides the following bars classes for use:

- *MDProgressBar*
- *Determinate*

- *Indeterminate*

## MDProgressBar

```
from kivy.lang import Builder

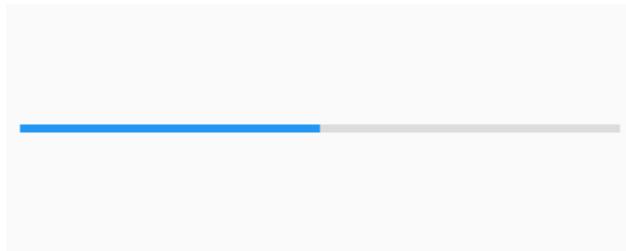
from kivymd.app import MDApp

KV = '''
BoxLayout:
 padding: "10dp"

 MDProgressBar:
 value: 50
'''

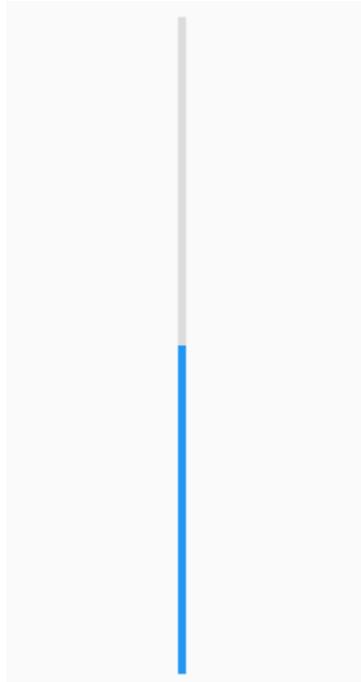
class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```



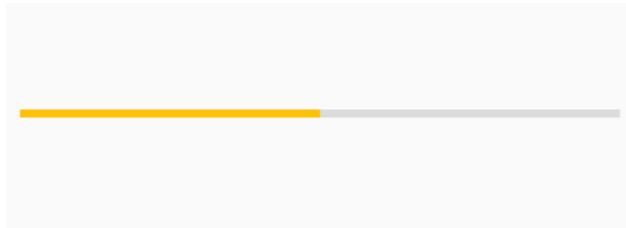
## Vertical orientation

```
MDProgressBar:
 orientation: "vertical"
 value: 50
```



### With custom color

```
MDProgressBar:
 value: 50
 color: app.theme_cls.accent_color
```



### Indeterminate

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp

KV = ''''
Screen:

 MDProgressBar:
 id: progress
 pos_hint: {"center_y": .6}
 type: "indeterminate"
```

(continues on next page)

(continued from previous page)

```

MDRaisedButton:
 text: "STOP" if app.state == "start" else "START"
 pos_hint: {"center_x": .5, "center_y": .45}
 on_press: app.state = "stop" if app.state == "start" else "start"
...

class Test(MDApp):
 state = StringProperty("stop")

 def build(self):
 return Builder.load_string(KV)

 def on_state(self, instance, value):
 {
 "start": self.root.ids.progress.start,
 "stop": self.root.ids.progress.stop,
 }.get(value)()
}

Test().run()

```

## Determinate

```

MDProgressBar:
 type: "determinate"
 running_duration: 1
 catching_duration: 1.5

```

## API - kivymd.uix.progressbar

**class kivymd.uix.progressbar.MDProgressBar(\*\*kwargs)**

Class for creating a progress bar widget.

See module documentation for more details.

**reversed**

Reverse the direction the progressbar moves.

*reversed* is an **BooleanProperty** and defaults to *False*.

**orientation**

Orientation of progressbar. Available options are: ‘horizontal’, ‘vertical’.

*orientation* is an **OptionProperty** and defaults to ‘horizontal’.

**color**

Progress bar color in rgba format.

`color` is an `ColorProperty` and defaults to `None`.

**running\_transition**

Running transition.

`running_transition` is an `StringProperty` and defaults to `'in_cubic'`.

**catching\_transition**

Catching transition.

`catching_transition` is an `StringProperty` and defaults to `'out_quart'`.

**running\_duration**

Running duration.

`running_duration` is an `NumericProperty` and defaults to `0.5`.

**catching\_duration**

Catching duration.

`catching_duration` is an `NumericProperty` and defaults to `0.8`.

**type**

Type of progressbar. Available options are: `'indeterminate'`, `'determinate'`.

`type` is an `OptionProperty` and defaults to `None`.

**start(self)**

Start animation.

**stop(self)**

Stop animation.

**running\_away(self, \*args)**

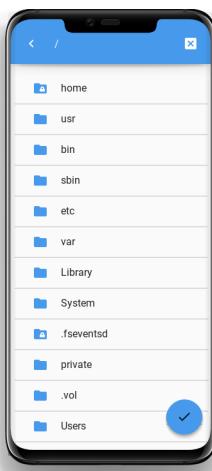
**catching\_up(self, \*args)**

## 2.3.16 File Manager

A simple manager for selecting directories and files.

### Usage

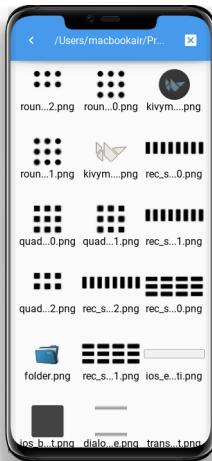
```
path = '/' # path to the directory that will be opened in the file manager
file_manager = MDFileManager(
 exit_manager=self.exit_manager, # function called when the user reaches directory ↴
 tree_root=tree_root,
 select_path=self.select_path, # function called when selecting a file/directory
)
file_manager.show(path)
```



**Warning:** Be careful! To use the / path on Android devices, you need special permissions. Therefore, you are likely to get an error.

Or with preview mode:

```
file_manager = MDFileManager(
 exit_manager=self.exit_manager,
 select_path=self.select_path,
 preview=True,
)
```



**Warning:** The *preview* mode is intended only for viewing images and will not display other types of files.

## Example

```
from kivy.core.window import Window
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.filemanager import MDFFileManager
from kivymd.toast import toast

KV = '''
BoxLayout:
 orientation: 'vertical'

 MDToolbar:
 title: "MDFFileManager"
 left_action_items: [['menu', lambda x: None]]
 elevation: 10

 FloatLayout:

 MDRoundFlatButton:
 text: "Open manager"
 icon: "folder"
 pos_hint: {'center_x': .5, 'center_y': .6}
 on_release: app.file_manager_open()
'''

class Example(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 Window.bind(on_keyboard=self.events)
 self.manager_open = False
 self.file_manager = MDFFileManager(
 exit_manager=self.exit_manager,
 select_path=self.select_path,
 preview=True,
)

 def build(self):
 return Builder.load_string(KV)

 def file_manager_open(self):
 self.file_manager.show('/') # output manager to the screen
 self.manager_open = True

 def select_path(self, path):
 "It will be called when you click on the file name
 or the catalog selection button.

 :type path: str;
```

(continues on next page)

(continued from previous page)

```
:param path: path to the selected directory or file;
"""

self.exit_manager()
toast(path)

def exit_manager(self, *args):
 "Called when the user reaches the root of the directory tree."

 self.manager_open = False
 self.file_manager.close()

def events(self, instance, keyboard, keycode, text, modifiers):
 "Called when buttons are pressed on the mobile device."

 if keyboard in (1001, 27):
 if self.manager_open:
 self.file_manager.back()
 return True
```

Example().run()

New in version 1.0.0.

Added a feature that allows you to show the available disks first, then the files contained in them. Works correctly on: *Windows, Linux, OSX, Android*. Not tested on *iOS*.

```
def file_manager_open(self):
 self.file_manager.show_disks()
```

## API - kivymd.uix.filemanager

**class kivymd.uix.filemanager.MDFileManager(\*\*kwargs)**  
 RelativeLayout class, see module documentation for more information.

### icon

The icon that will be used on the directory selection button.

*icon* is an **StringProperty** and defaults to *check*.

### icon\_folder

The icon that will be used for folder icons when using `preview = True`.

*icon* is an **StringProperty** and defaults to *check*.

### exit\_manager

Function called when the user reaches directory tree root.

*exit\_manager* is an **ObjectProperty** and defaults to *lambda x: None*.

### select\_path

Function, called when selecting a file/directory.

*select\_path* is an **ObjectProperty** and defaults to *lambda x: None*.

**ext**

List of file extensions to be displayed in the manager. For example, `['.py', '.kv']` - will filter out all files, except python scripts and Kv Language.

`ext` is an [ListProperty](#) and defaults to `[]`.

**search**

It can take the values ‘all’ ‘dirs’ ‘files’ - display only directories or only files or both them. By default, it displays folders, and files. Available options are: ‘all’, ‘dirs’, ‘files’.

`search` is an [OptionProperty](#) and defaults to `all`.

**current\_path**

Current directory.

`current_path` is an [StringProperty](#) and defaults to `/`.

**use\_access**

Show access to files and directories.

`use_access` is an [BooleanProperty](#) and defaults to `True`.

**preview**

Shows only image previews.

`preview` is an [BooleanProperty](#) and defaults to `False`.

**show\_hidden\_files**

Shows hidden files.

`show_hidden_files` is an [BooleanProperty](#) and defaults to `False`.

**sort\_by**

It can take the values ‘nothing’ ‘name’ ‘date’ ‘size’ ‘type’ - sorts files by option By default, sort by name. Available options are: ‘nothing’, ‘name’, ‘date’, ‘size’, ‘type’.

`sort_by` is an [OptionProperty](#) and defaults to `name`.

**sort\_by\_desc**

Sort by descending.

`sort_by_desc` is an [BooleanProperty](#) and defaults to `False`.

**selector**

It can take the values ‘any’ ‘file’ ‘folder’ ‘multi’ By default, any. Available options are: ‘any’, ‘file’, ‘folder’, ‘multi’.

`selector` is an [OptionProperty](#) and defaults to `any`.

**selection**

Contains the list of files that are currently selected.

`selection` is a read-only [ListProperty](#) and defaults to `[]`.

**show\_disks(self)**

**show(self, path)**

Forms the body of a directory tree.

**Parameters** `path` – The path to the directory that will be opened in the file manager.

**get\_access\_string(self, path)**

**get\_content(self)**

Returns a list of the type [[Folder List], [file list]].

**close(self)**  
 Closes the file manager window.

**select\_dir\_or\_file(self, path, widget)**  
 Called by tap on the name of the directory or file.

**back(self)**  
 Returning to the branch down in the directory tree.

**select\_directory\_on\_press\_button(self, \*args)**  
 Called when a click on a floating button.

### 2.3.17 CircularLayout

CircularLayout is a special layout that places widgets around a circle.

#### MDCircularLayout

##### Usage

```
from kivy.lang.builder import Builder
from kivy.uix.label import Label

from kivymd.app import MDApp

kv = '''
Screen:
 MDCircularLayout:
 id: container
 pos_hint: {"center_x": .5, "center_y": .5}
 row_spacing: min(self.size)*0.1
'''

class Main(MDApp):
 def build(self):
 return Builder.load_string(kv)

 def on_start(self):
 for x in range(1, 49):
 self.root.ids.container.add_widget(
 Label(text=f"{x}", color=[0, 0, 0, 1])
)

Main().run()
```

**API - kivymd.uix.circularlayout****class kivymd.uix.circularlayout.MDCircularLayout(\*\*kwargs)**

Float layout class. See module documentation for more information.

**degree\_spacing**

The space between children in degree.

*degree\_spacing* is an [NumericProperty](#) and defaults to *30*.**circular\_radius**Radius of circle. Radius will be the greatest value in the layout if *circular\_radius* if not specified.*circular\_radius* is an [NumericProperty](#) and defaults to *None*.**start\_from**

The positon of first child in degree.

*start\_from* is an [NumericProperty](#) and defaults to *60*.**max\_degree**

Maximum range in degree allowed for each row of widgets before jumping to the next row.

*max\_degree* is an [NumericProperty](#) and defaults to *360*.**circular\_padding**

Padding between outer widgets and the edge of the biggest circle.

*circular\_padding* is an [NumericProperty](#) and defaults to *25dp*.**row\_spacing**

Space between each row of widget.

*row\_spacing* is an [NumericProperty](#) and defaults to *50dp*.**clockwise**

Direction of widgets in circular direction.

*clockwise* is an [BooleanProperty](#) and defaults to *True*.**do\_layout(self, \*largs, \*\*kwargs)**This function is called when a layout is called by a trigger. If you are writing a new Layout subclass, don't call this function directly but use *\_trigger\_layout()* instead.The function is by default called *before* the next frame, therefore the layout isn't updated immediately. Anything depending on the positions of e.g. children should be scheduled for the next frame.

New in version 1.0.8.

**remove\_widget(self, widget, \*\*kwargs)**

Remove a widget from the children of this widget.

**Parameters****widget: Widget** Widget to remove from our children list.

```
>>> from kivy.uix.button import Button
>>> root = Widget()
>>> button = Button()
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

**get\_angle(self, pos)**

Returns the angle of given pos

### 2.3.18 Refresh Layout

#### Example

```
from kivymd.app import MDApp
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.factory import Factory
from kivy.properties import StringProperty

from kivymd.uix.button import MDIconButton
from kivymd.icon_definitions import md_icons
from kivymd.uix.list import ILeftBodyTouch, OneLineIconListItem
from kivymd.theming import ThemeManager
from kivymd.utils import asynckivy

Builder.load_string('''
<ItemForList>
 text: root.text

 IconLeftSampleWidget:
 icon: root.icon

<Example@FloatLayout>

 BoxLayout:
 orientation: 'vertical'

 MDToolbar:
 title: app.title
 md_bg_color: app.theme_cls.primary_color
 background_palette: 'Primary'
 elevation: 10
 left_action_items: [['menu', lambda x: x]]

 MDScrollViewRefreshLayout:
 id: refresh_layout
 refresh_callback: app.refresh_callback
 root_layout: root

 MDGridLayout:
 id: box
 adaptive_height: True
 cols: 1
 ''')

class IconLeftSampleWidget(ILeftBodyTouch, MDIconButton):
 pass
```

(continues on next page)

(continued from previous page)

```
class ItemForList(OneLineIconListItem):
 icon = StringProperty()

class Example(MDApp):
 title = 'Example Refresh Layout'
 screen = None
 x = 0
 y = 15

 def build(self):
 self.screen = Factory.Example()
 self.set_list()

 return self.screen

 def set_list(self):
 async def set_list():
 names_icons_list = list(md_icons.keys())[self.x:self.y]
 for name_icon in names_icons_list:
 await asynckivy.sleep(0)
 self.screen.ids.box.add_widget(
 ItemForList(icon=name_icon, text=name_icon))
 asynckivy.start(set_list())

 def refresh_callback(self, *args):
 """A method that updates the state of your application
 while the spinner remains on the screen."""

 def refresh_callback(interval):
 self.screen.ids.box.clear_widgets()
 if self.x == 0:
 self.x, self.y = 15, 30
 else:
 self.x, self.y = 0, 15
 self.set_list()
 self.screen.ids.refresh_layout.refresh_done()
 self.tick = 0

 Clock.schedule_once(refresh_callback, 1)

Example().run()
```

## API - kivymd.uix.refreshlayout

```
class kivymd.uix.refreshlayout.MDScrollViewRefreshLayout(**kwargs)
```

ScrollView class. See module documentation for more information.

### Events

*on\_scroll\_start* Generic event fired when scrolling starts from touch.

*on\_scroll\_move* Generic event fired when scrolling move from touch.

*on\_scroll\_stop* Generic event fired when scrolling stops from touch.

Changed in version 1.9.0: *on\_scroll\_start*, *on\_scroll\_move* and *on\_scroll\_stop* events are now dispatched when scrolling to handle nested ScrollViews.

Changed in version 1.7.0: *auto\_scroll*, *scroll\_friction*, *scroll\_moves*, *scroll\_stoptime*' has been deprecated, use *:attr: effect\_cls* instead.

### root\_layout

The spinner will be attached to this layout.

### on\_touch\_up(self, \*args)

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

### refresh\_done(self)

```
class kivymd.uix.refreshlayout.RefreshSpinner(**kwargs)
```

Float layout class. See module documentation for more information.

### spinner\_color

### start\_anim\_spinner(self)

### hide\_anim\_spinner(self)

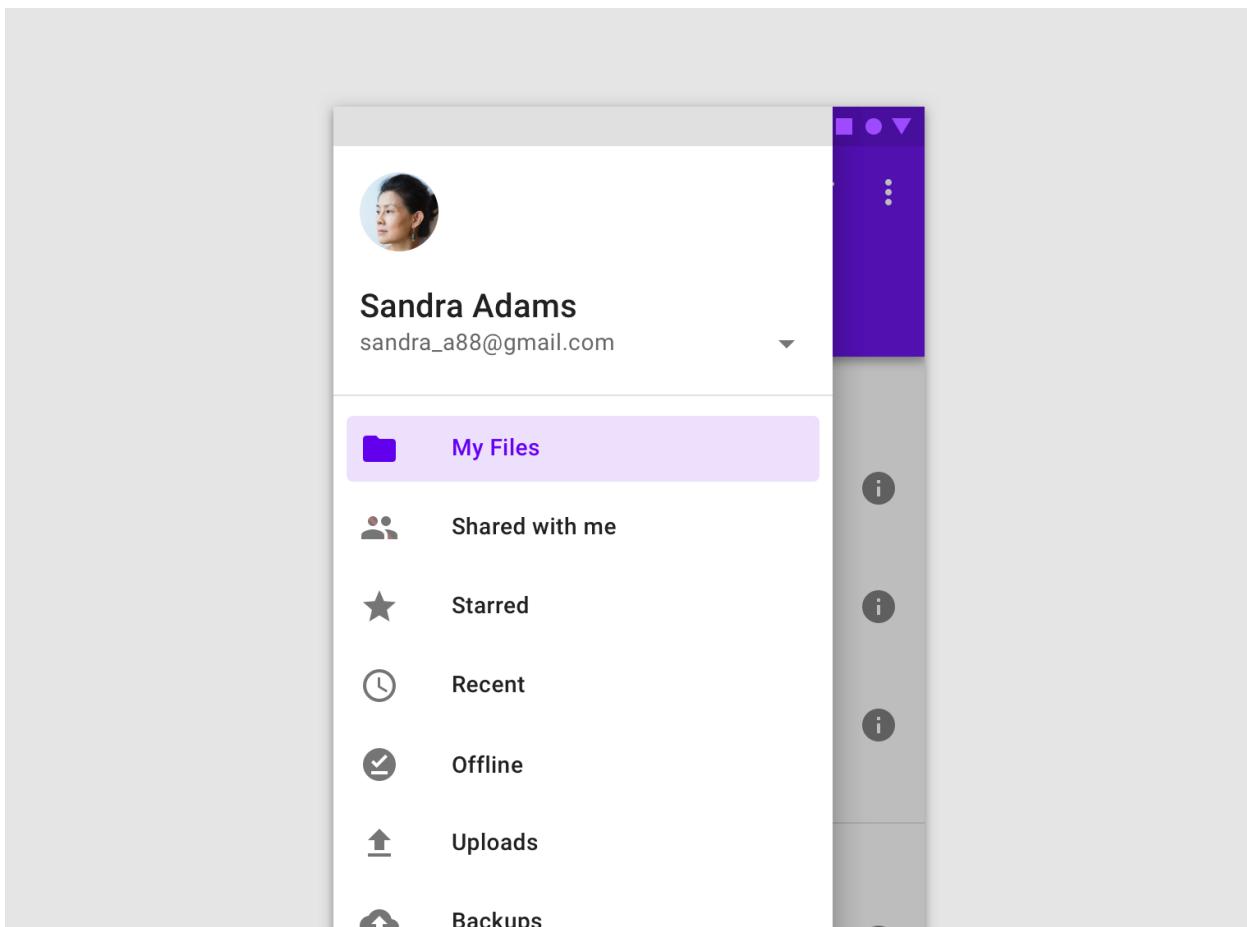
### set\_spinner(self, \*args)

## 2.3.19 Navigation Drawer

### See also:

Material Design spec, Navigation drawer

Navigation drawers provide access to destinations in your app.



When using the class `MDNavigationDrawer` skeleton of your KV markup should look like this:

**Root:**

**MDNavigationLayout:**

**ScreenManager:**

**Screen\_1:**

**Screen\_2:**

**MDNavigationDrawer:**

```
This custom rule should implement what will be appear in your
→MDNavigationDrawer
 ContentNavigationDrawer
```

A simple example:

```
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
```

(continues on next page)

(continued from previous page)

```

from kivymd.app import MDApp

KV = """
Screen:

 MDNavigationLayout:

 ScreenManager:

 Screen:

 BoxLayout:
 orientation: 'vertical'

 MDToolbar:
 title: "Navigation Drawer"
 elevation: 10
 left_action_items: [['menu', lambda x: nav_drawer.set_state("open"
→")]]

 Widget:

 MDNavigationDrawer:
 id: nav_drawer

 ContentNavigationDrawer:
 ...

```

---

```

class ContentNavigationDrawer(BoxLayout):
 pass

class TestNavigationDrawer(MDApp):
 def build(self):
 return Builder.load_string(KV)

TestNavigationDrawer().run()

```

---

**Note:** `MDNavigationDrawer` is an empty `MDCard` panel.

---

Let's extend the `ContentNavigationDrawer` class from the above example and create content for our `MDNavigationDrawer` panel:

```

Menu item in the DrawerList list.
<ItemDrawer>
 theme_text_color: "Custom"

```

(continues on next page)

(continued from previous page)

```
on_release: self.parent.set_color_item(self)
```

```
IconLeftWidget:
 id: icon
 icon: root.icon
 theme_text_color: "Custom"
 text_color: root.text_color
```

```
class ItemDrawer(OneLineIconListItem):
 icon = StringProperty()
```



Top of ContentNavigationDrawer and DrawerList for menu items:

```
<ContentNavigationDrawer>:
 orientation: "vertical"
 padding: "8dp"
 spacing: "8dp"

 AnchorLayout:
 anchor_x: "left"
 size_hint_y: None
 height: avatar.height

 Image:
 id: avatar
 size_hint: None, None
 size: "56dp", "56dp"
 source: "kivymd.png"

 MDLabel:
 text: "KivyMD library"
 font_style: "Button"
 size_hint_y: None
 height: self.texture_size[1]

 MDLabel:
 text: "kivydevelopment@gmail.com"
 font_style: "Caption"
 size_hint_y: None
 height: self.texture_size[1]

 ScrollView:

 DrawerList:
 id: md_list
```

```
class ContentNavigationDrawer(BoxLayout):
 pass
```

(continues on next page)

(continued from previous page)

```
class DrawerList(ThemableBehavior, MDList):
 def set_color_item(self, instance_item):
 "Called when tap on a menu item."

 # Set the color of the icon and text for the menu item.
 for item in self.children:
 if item.text_color == self.theme_cls.primary_color:
 item.text_color = self.theme_cls.text_color
 break
 instance_item.text_color = self.theme_cls.primary_color
```

**KIVYMD LIBRARY**

kivydevelopment@gmail.com

Create a menu list for ContentNavigationDrawer:

```
def on_start(self):
 icons_item = {
 "folder": "My files",
 "account-multiple": "Shared with me",
 "star": "Starred",
 "history": "Recent",
 "checkbox-marked": "Shared with me",
 "upload": "Upload",
 }
 for icon_name in icons_item.keys():
 self.root.ids.content_drawer.ids.md_list.add_widget(
 ItemDrawer(icon=icon_name, text=icons_item[icon_name])
)
```

### Switching screens in the ScreenManager and using the common MDToolbar

```
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import ObjectProperty

from kivymd.app import MDApp

KV = '''
<ContentNavigationDrawer>:
```

(continues on next page)

(continued from previous page)

```
ScrollView:
```

```
 MDList:
```

```
 OneLineListItem:
 text: "Screen 1"
 on_press:
 root.nav_drawer.set_state("close")
 root.screen_manager.current = "scr 1"
```

```
 OneLineListItem:
 text: "Screen 2"
 on_press:
 root.nav_drawer.set_state("close")
 root.screen_manager.current = "scr 2"
```

```
Screen:
```

```
 MDToolbar:
 id: toolbar
 pos_hint: {"top": 1}
 elevation: 10
 title: "MDNavigationDrawer"
 left_action_items: [["menu", lambda x: nav_drawer.set_state("open")]]
```

```
 MDNavigationLayout:
 x: toolbar.height
```

```
 ScreenManager:
 id: screen_manager
```

```
 Screen:
 name: "scr 1"
```

```
 MDLabel:
 text: "Screen 1"
 halign: "center"
```

```
 Screen:
 name: "scr 2"
```

```
 MDLabel:
 text: "Screen 2"
 halign: "center"
```

```
 MDNavigationDrawer:
 id: nav_drawer
```

```
 ContentNavigationDrawer:
 screen_manager: screen_manager
 nav_drawer: nav_drawer
```

(continues on next page)

(continued from previous page)

```
...
class ContentNavigationDrawer(BoxLayout):
 screen_manager = ObjectProperty()
 nav_drawer = ObjectProperty()

class TestNavigationDrawer(MDApp):
 def build(self):
 return Builder.load_string(KV)

TestNavigationDrawer().run()
```

## NavigationDrawer with type standard

You can use the `standard` behavior type for the NavigationDrawer:

```
MDNavigationDrawer:
 type: "standard"
```

### See also:

Full example of Components-Navigation-Drawer

### API - kivymd.uix.navigationdrawer

```
class kivymd.uix.navigationdrawer.MDNavigationLayout(**kwargs)
 Float layout class. See module documentation for more information.

 update_pos(self, *args)
 add_scrim(self, widget)
 update_scrim_rectangle(self, *args)
 add_widget(self, widget, index=0, canvas=None)
 Only two layouts are allowed: ScreenManager and MDNavigationDrawer.
```

```
class kivymd.uix.navigationdrawer.MDNavigationDrawer(**kwargs)
 FakeRectangularElevationBehavior is a shadow mockup for widgets. Improves performance using cached images inside kivymd.images dir
```

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
 ThemableBehavior,
 FakeCircularElevationBehavior,
 SpecificBackgroundColorBehavior,
 # here you add the other front end classes for the widget front_end,
):
 [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

*FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
 MDCard,
 FakeRectangularElevationBehavior
):
 [...]
```

---

**Note:** About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

---

#### **type**

Type of drawer. Modal type will be on top of screen. Standard type will be at left or right of screen. Also it automatically disables *close\_on\_click* and *enable\_swiping* to prevent closing drawer for standard type.

*type* is a [OptionProperty](#) and defaults to *modal*.

#### **anchor**

Anchoring screen edge for drawer. Set it to '*right*' for right-to-left languages. Available options are: '*left*', '*right*'.

*anchor* is a [OptionProperty](#) and defaults to *left*.

#### **close\_on\_click**

Close when click on scrim or keyboard escape. It automatically sets to False for "standard" type.

*close\_on\_click* is a [BooleanProperty](#) and defaults to *True*.

#### **state**

Indicates if panel closed or opened. Sets after *status* change. Available options are: '*close*', '*open*'.

*state* is a [OptionProperty](#) and defaults to '*close*'.

#### **status**

Detailed state. Sets before *state*. Bind to *state* instead of *status*. Available options are: '*closed*', '*opening\_with\_swipe*', '*opening\_with\_animation*', '*opened*', '*closing\_with\_swipe*', '*closing\_with\_animation*'.

*status* is a [OptionProperty](#) and defaults to '*closed*'.

#### **open\_progress**

Percent of visible part of side panel. The percent is specified as a floating point number in the range 0-1. 0.0 if panel is closed and 1.0 if panel is opened.

`open_progress` is a `NumericProperty` and defaults to `0.0`.

#### `enable_swiping`

Allow to open or close navigation drawer with swipe. It automatically sets to False for “standard” type.

`enable_swiping` is a `BooleanProperty` and defaults to `True`.

#### `swipe_distance`

The distance of the swipe with which the movement of navigation drawer begins.

`swipe_distance` is a `NumericProperty` and defaults to `10`.

#### `swipe_edge_width`

The size of the area in px inside which should start swipe to drag navigation drawer.

`swipe_edge_width` is a `NumericProperty` and defaults to `20`.

#### `scrim_color`

Color for scrim. Alpha channel will be multiplied with `_scrim_alpha`. Set fourth channel to 0 if you want to disable scrim.

`scrim_color` is a `ColorProperty` and defaults to `[0, 0, 0, 0.5]`.

#### `scrim_alpha_transition`

The name of the animation transition type to use for changing `scrim_alpha`.

`scrim_alpha_transition` is a `StringProperty` and defaults to ‘`linear`’.

#### `opening_transition`

The name of the animation transition type to use when animating to the `state ‘open’`.

`opening_transition` is a `StringProperty` and defaults to ‘`out_cubic`’.

#### `opening_time`

The time taken for the panel to slide to the `state ‘open’`.

`opening_time` is a `NumericProperty` and defaults to `0.2`.

#### `closing_transition`

The name of the animation transition type to use when animating to the `state ‘close’`.

`closing_transition` is a `StringProperty` and defaults to ‘`out_sine`’.

#### `closing_time`

The time taken for the panel to slide to the `state ‘close’`.

`closing_time` is a `NumericProperty` and defaults to `0.2`.

#### `set_state(self, new_state='toggle', animation=True)`

Change state of the side panel. New\_state can be one of “`toggle`”, “`open`” or “`close`”.

#### `update_status(self, *_)`

#### `get_dist_from_side(self, x)`

#### `on_touch_down(self, touch)`

Receive a touch down event.

#### Parameters

`touch: MotionEvent class` Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

**Returns** bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

**on\_touch\_move(self, touch)**

Receive a touch move event. The touch is in parent coordinates.

See [on\\_touch\\_down\(\)](#) for more information.

**on\_touch\_up(self, touch)**

Receive a touch up event. The touch is in parent coordinates.

See [on\\_touch\\_down\(\)](#) for more information.

**on\_radius(self, instance, value)**

**on\_type(self, \*args)**

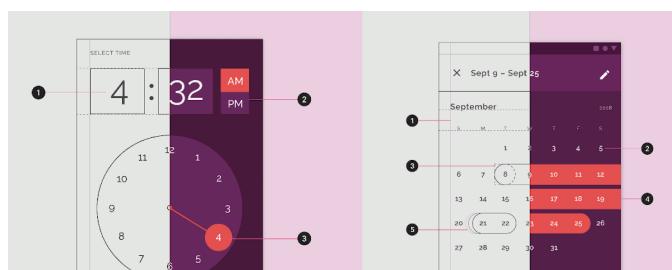
## 2.3.20 Pickers

### See also:

[Material Design spec, Time picker](#)

[Material Design spec, Date picker](#)

**Includes date, time and color picker.**



KivyMD provides the following classes for use:

- [MDTimePicker](#)
- [MDDatePicker](#)
- [MDThemePicker](#)

### MDTimePicker

#### Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.picker import MDTimePicker

KV = '''
MDFloatLayout:

 MDRaisedButton:
```

(continues on next page)

(continued from previous page)

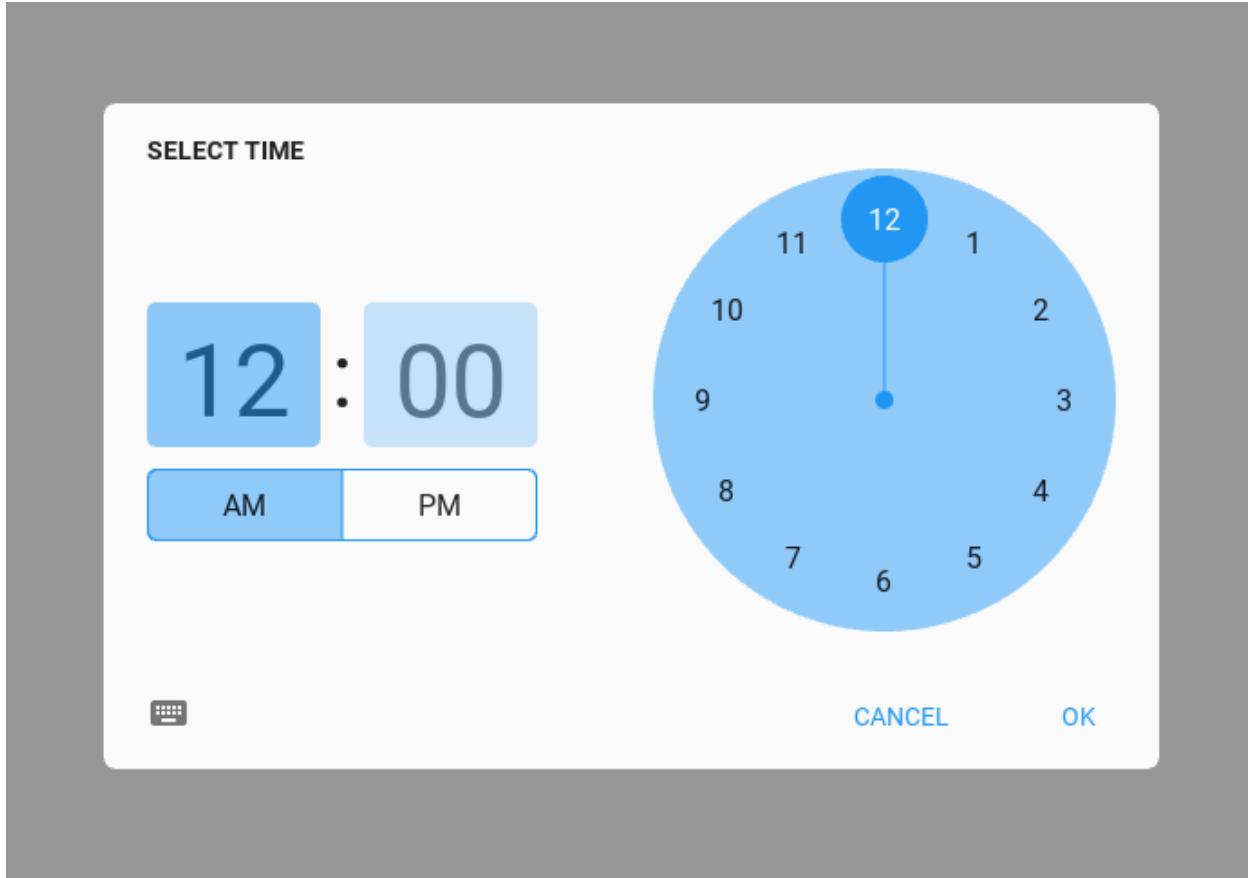
```
text: "Open time picker"
pos_hint: {'center_x': .5, 'center_y': .5}
on_release: app.show_time_picker()
...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def show_time_picker(self):
 "Open time picker dialog."

 time_dialog = MDTimePicker()
 time_dialog.open()

Test().run()
```



## Binding method returning set time

```
def show_time_picker(self):
 time_dialog = MDTimePicker()
 time_dialog.bind(time=self.get_time)
 time_dialog.open()

def get_time(self, instance, time):
 """
 The method returns the set time.

 :type instance: <kivymd.uix.picker.MDTimePicker object>
 :type time: <class 'datetime.time'>
 """

 return time
```

## Open time dialog with the specified time

Use the `set_time` method of the class.

```
def show_time_picker(self):
 from datetime import datetime

 # Must be a datetime object
 previous_time = datetime.strptime("03:20:00", "%H:%M:%S").time()
 time_dialog = MDTimePicker()
 time_dialog.set_time(previous_time)
 time_dialog.open()
```

---

**Note:** For customization of the `MDTimePicker` class, see the documentation in the `BaseDialogPicker` class.

---

## MDDatePicker

**Warning:** The widget is under testing. Therefore, we would be grateful if you would let us know about the bugs found.

## Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.picker import MDDatePicker

KV = '''
MDFloatLayout:
```

(continues on next page)

(continued from previous page)

```
MDToolbar:
 title: "MDDatePicker"
 pos_hint: {"top": 1}
 elevation: 10

MDRaisedButton:
 text: "Open time picker"
 pos_hint: {'center_x': .5, 'center_y': .5}
 on_release: app.show_date_picker()
...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def on_save(self, instance, value, date_range):
 """
 Events called when the "OK" dialog box button is clicked.

 :type instance: <kivymd.uix.picker.MDDatePicker object>;
 :param value: selected date;
 :type value: <class 'datetime.date'>;
 :param date_range: list of 'datetime.date' objects in the selected range;
 :type date_range: <class 'list'>;
 """

 print(instance, value, date_range)

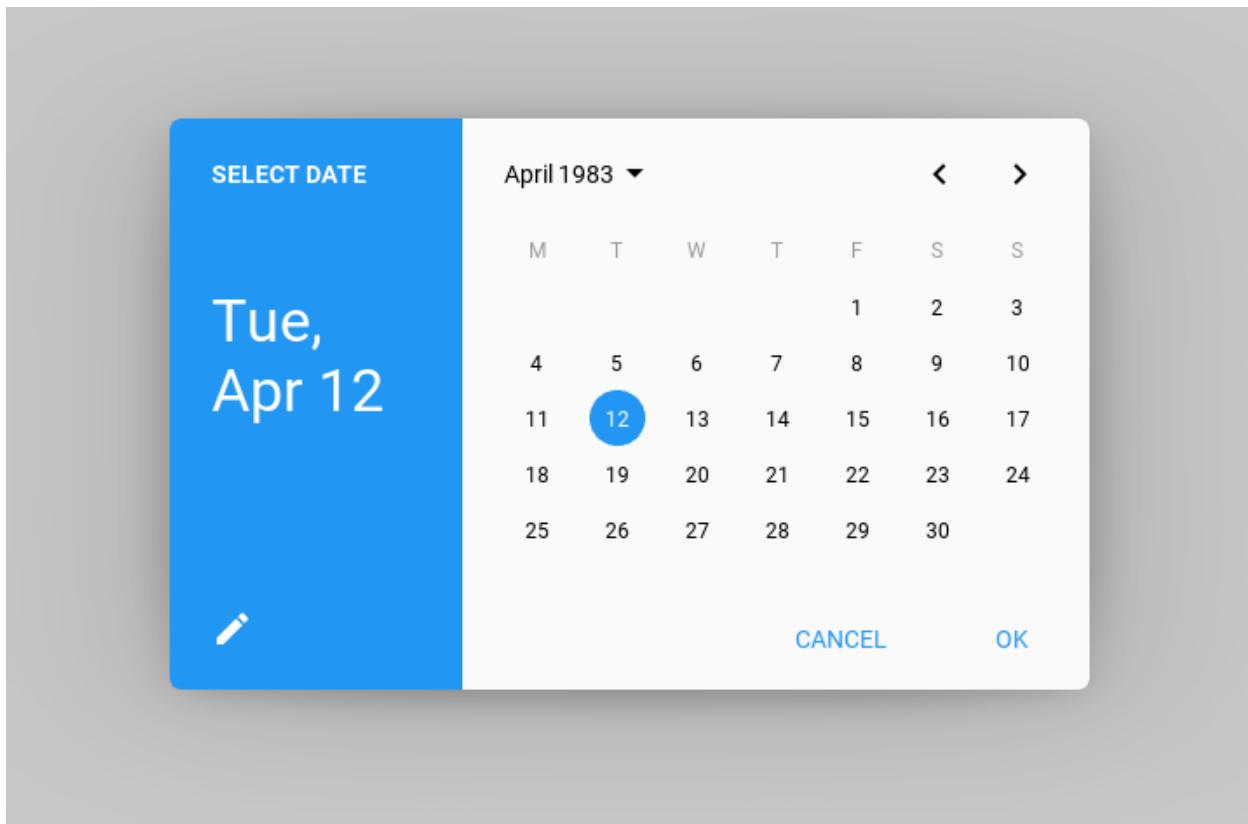
 def on_cancel(self, instance, value):
 "Events called when the "CANCEL" dialog box button is clicked."

 def show_date_picker(self):
 date_dialog = MDDatePicker()
 date_dialog.bind(on_save=self.on_save, on_cancel=self.on_cancel)
 date_dialog.open()

Test().run()
```

### Open date dialog with the specified date

```
def show_date_picker(self):
 date_dialog = MDDDatePicker(year=1983, month=4, day=12)
 date_dialog.open()
```



You can set the time interval from and to the set date. All days of the week that are not included in this range will have the status *disabled*.

```
def show_date_picker(self):
 date_dialog = MDDDatePicker(
 min_date=datetime.date(2021, 2, 15),
 max_date=datetime.date(2021, 3, 27),
)
 date_dialog.open()
```

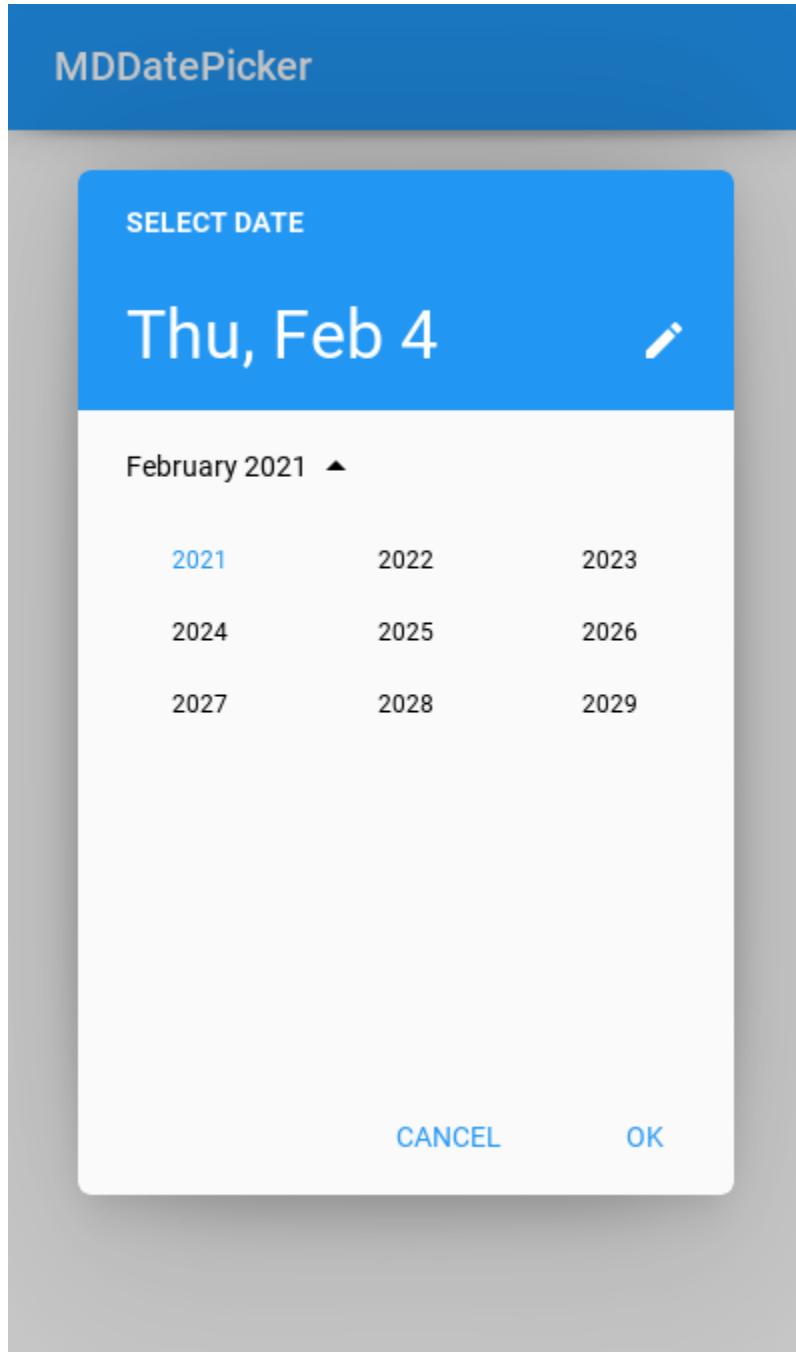
The range of available dates can be changed in the picker dialog:

## Select year

**Warning:** The list of years when opening is not automatically set to the current year.

You can set the range of years using the `min_year` and `max_year` attributes:

```
def show_date_picker(self):
 date_dialog = MDDatePicker(min_year=2021, max_year=2030)
 date_dialog.open()
```



## Set and select a date range

```
def show_date_picker(self):
 date_dialog = MDDatePicker(mode="range")
 date_dialog.open()
```

## MDThemePicker

```
def show_theme_picker(self):
 theme_dialog = MDThemePicker()
 theme_dialog.open()
```

## API - kivymd.uix.picker

**class kivymd.uix.picker.BaseDialogPicker(\*\*kwargs)**  
Base class for `MDDatePicker` and `MDTimePicker` classes.

### Events

***on\_save*** Events called when the “OK” dialog box button is clicked.

***on\_cancel*** Events called when the “CANCEL” dialog box button is clicked.

### **title\_input**

Dialog title for input date.

***title\_input*** is an `StringProperty` and defaults to *INPUT DATE*.

### **title**

Dialog title for select date.

***title*** is an `StringProperty` and defaults to *SELECT DATE*.

### **radius**

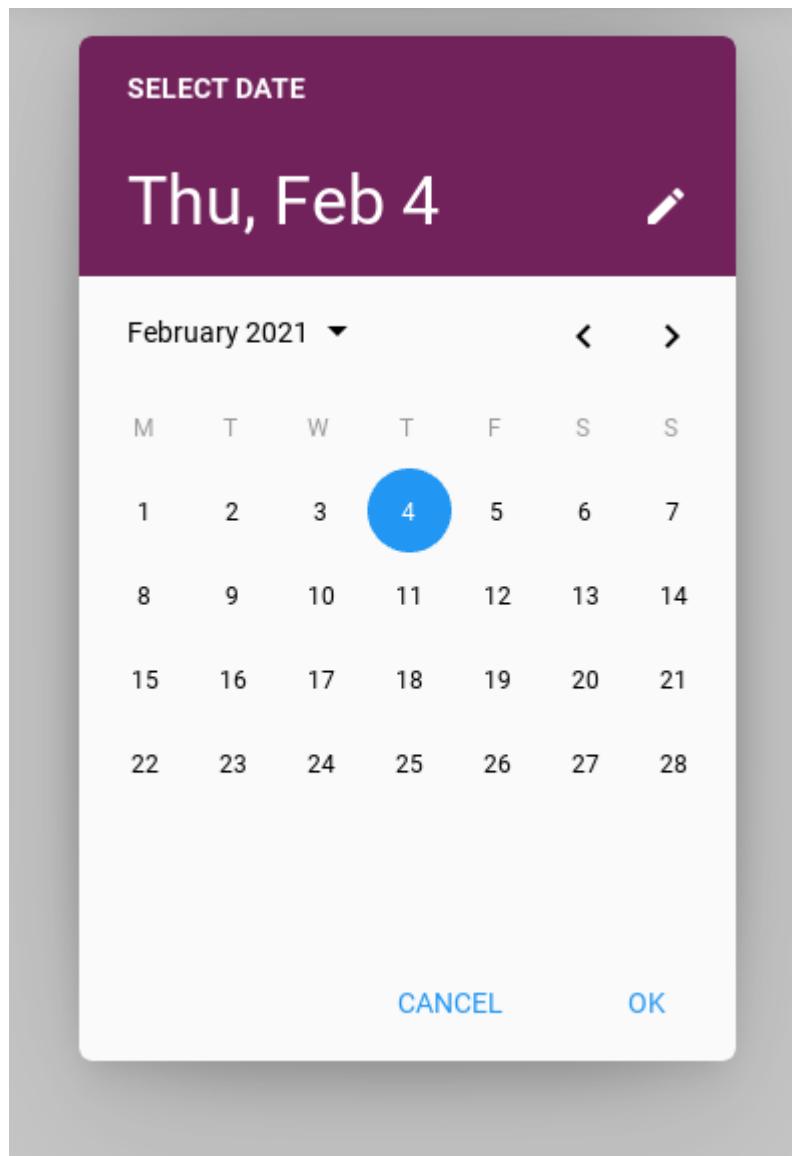
Radius list for the four corners of the dialog.

***radius*** is an `ListProperty` and defaults to [7, 7, 7, 7].

### **primary\_color**

Background color of toolbar.

```
MDDatePicker(primary_color=get_color_from_hex("#72225b"))
```

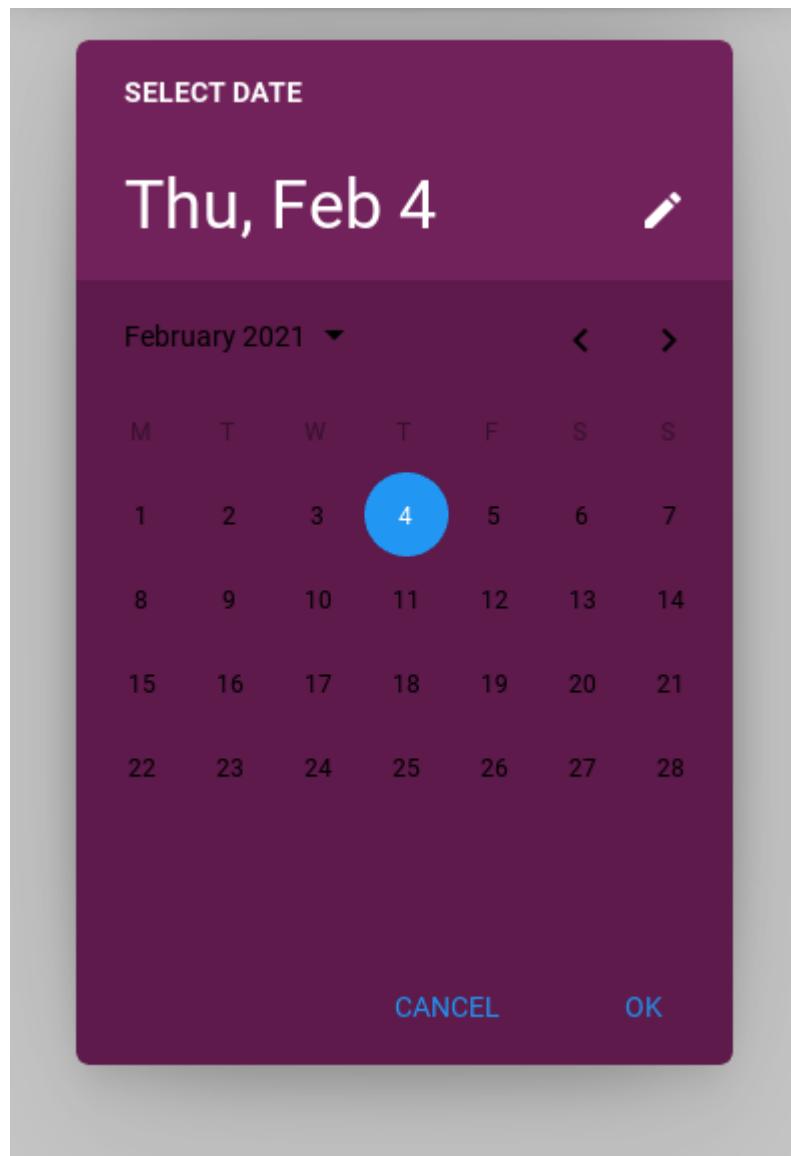


*primary\_color* is an `ColorProperty` and defaults to `None`.

#### **accent\_color**

Background color of calendar/clock face.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
)
```

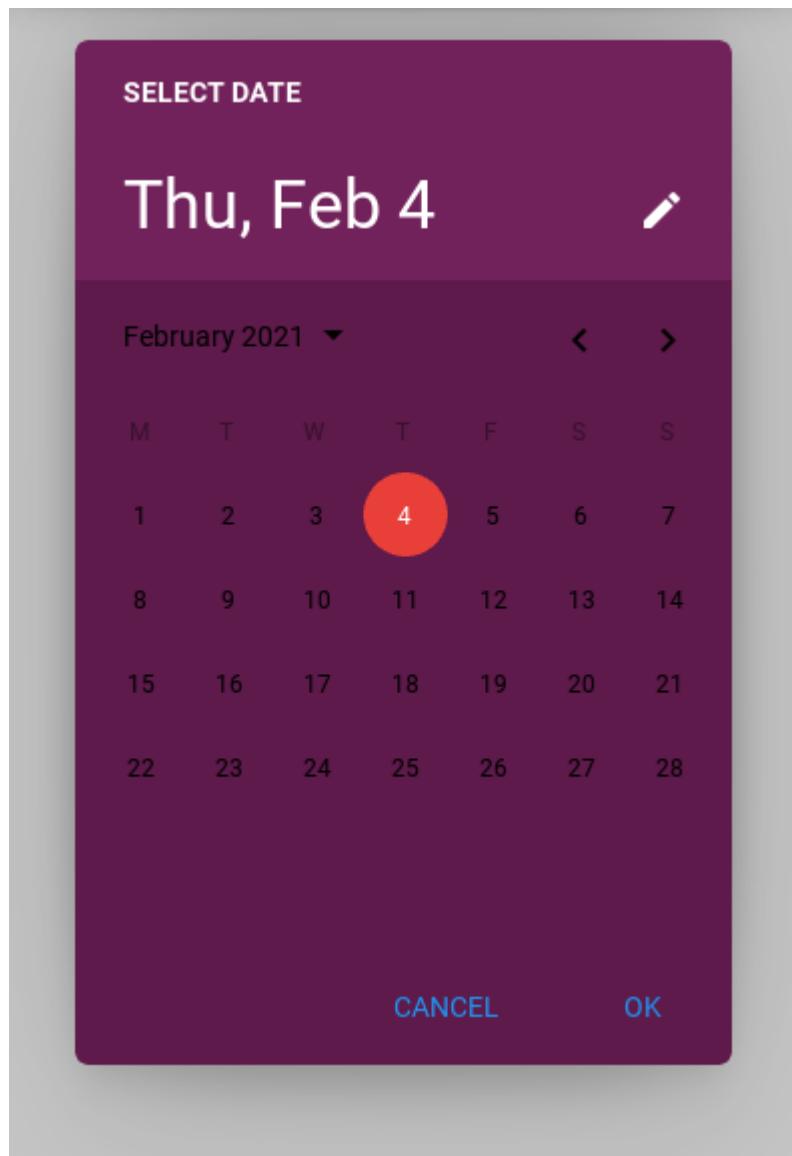


`accent_color` is an [ColorProperty](#) and defaults to *None*.

#### **selector\_color**

Background color of the selected day of the month or hour.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
)
```

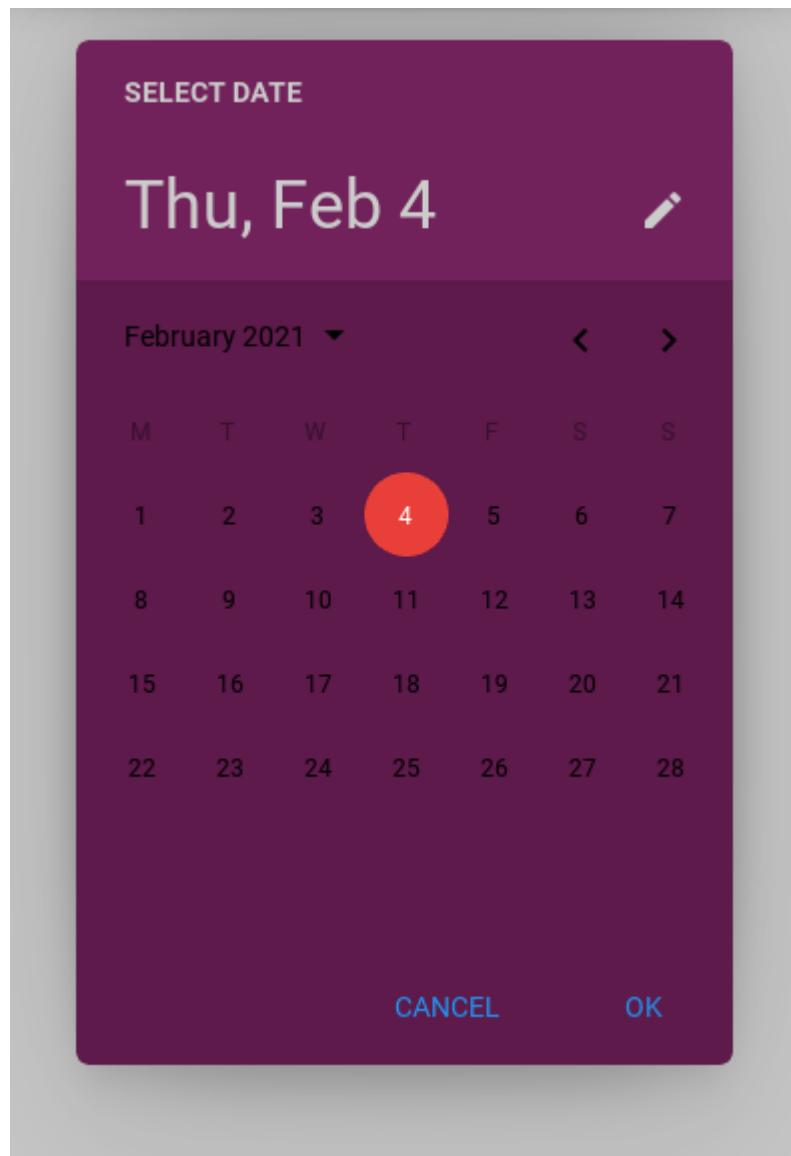


`selector_color` is an `ColorProperty` and defaults to `None`.

#### `text_toolbar_color`

Color of labels for text on a toolbar.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
)
```

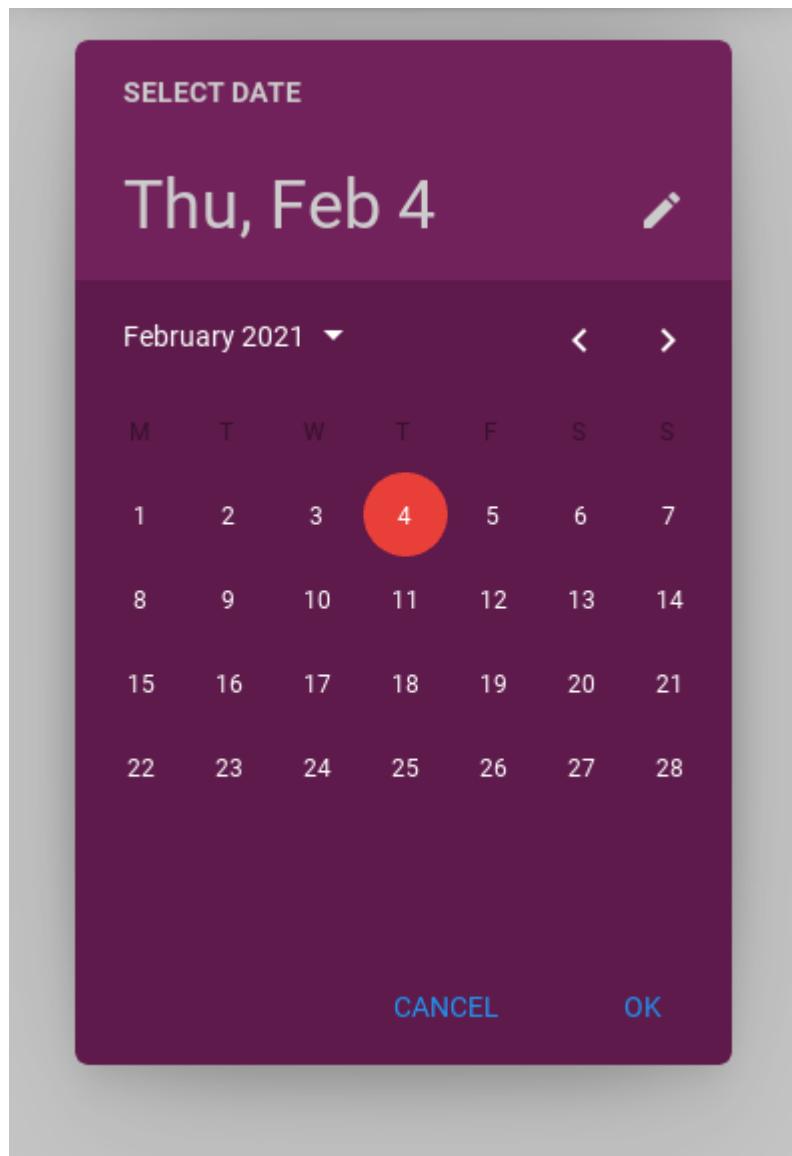


`text_toolbar_color` is an `ColorProperty` and defaults to `None`.

#### **text\_color**

Color of text labels in calendar/clock face.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffffff",
)
```

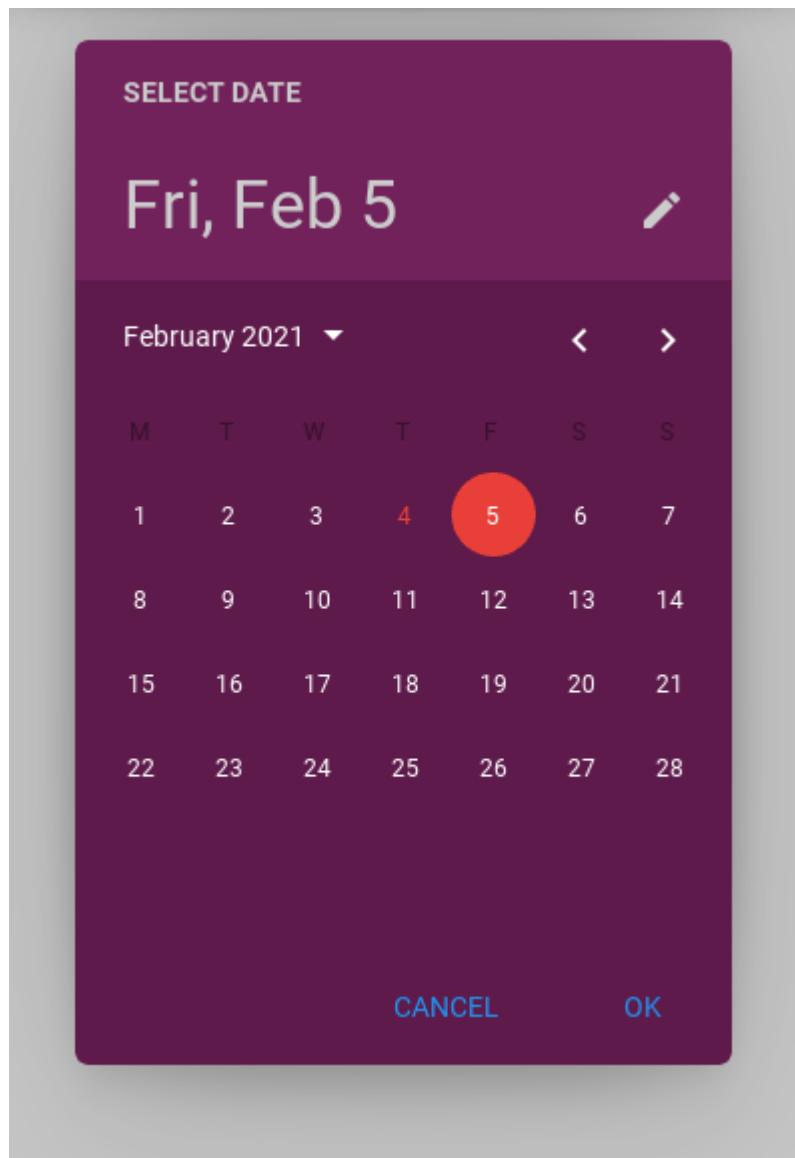


`text_color` is an [ColorProperty](#) and defaults to *None*.

#### `text_current_color`

Color of the text of the current day of the month/hour.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffff",
 text_current_color=get_color_from_hex("#e93f39"),
)
```

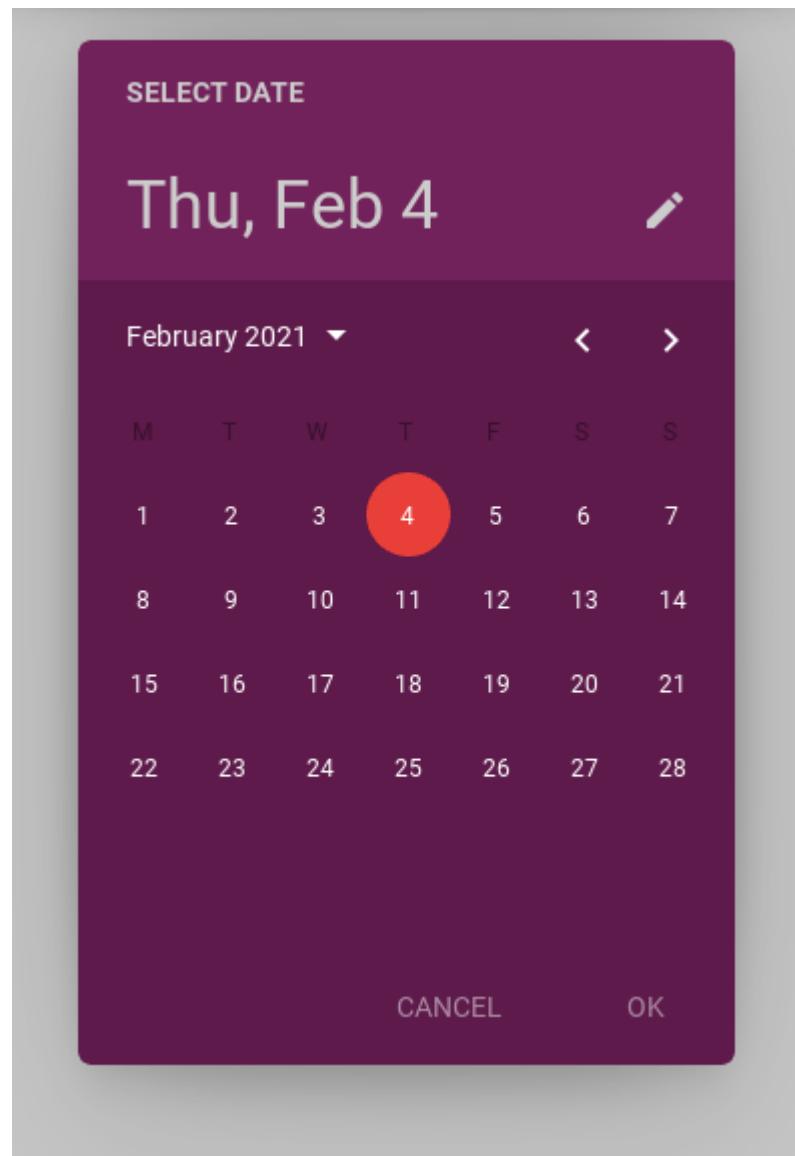


`text_current_color` is an `ColorProperty` and defaults to `None`.

#### **text\_button\_color**

Text button color.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffffff",
 text_current_color=get_color_from_hex("#e93f39"),
 text_button_color=(1, 1, 1, .5),
)
```

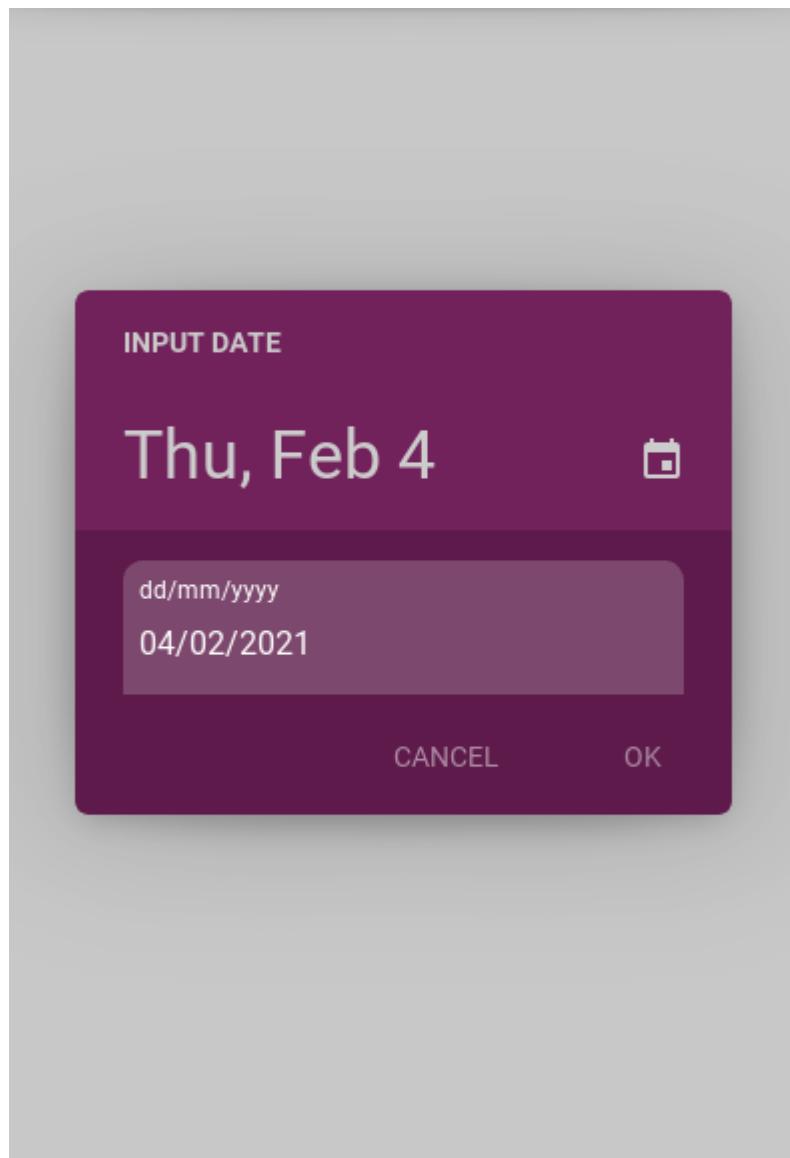


`text_button_color` is an `ColorProperty` and defaults to `None`.

#### `input_field_background_color`

Background color of input fields.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffff",
 text_current_color=get_color_from_hex("#e93f39"),
 input_field_background_color=(1, 1, 1, 0.2),
)
```



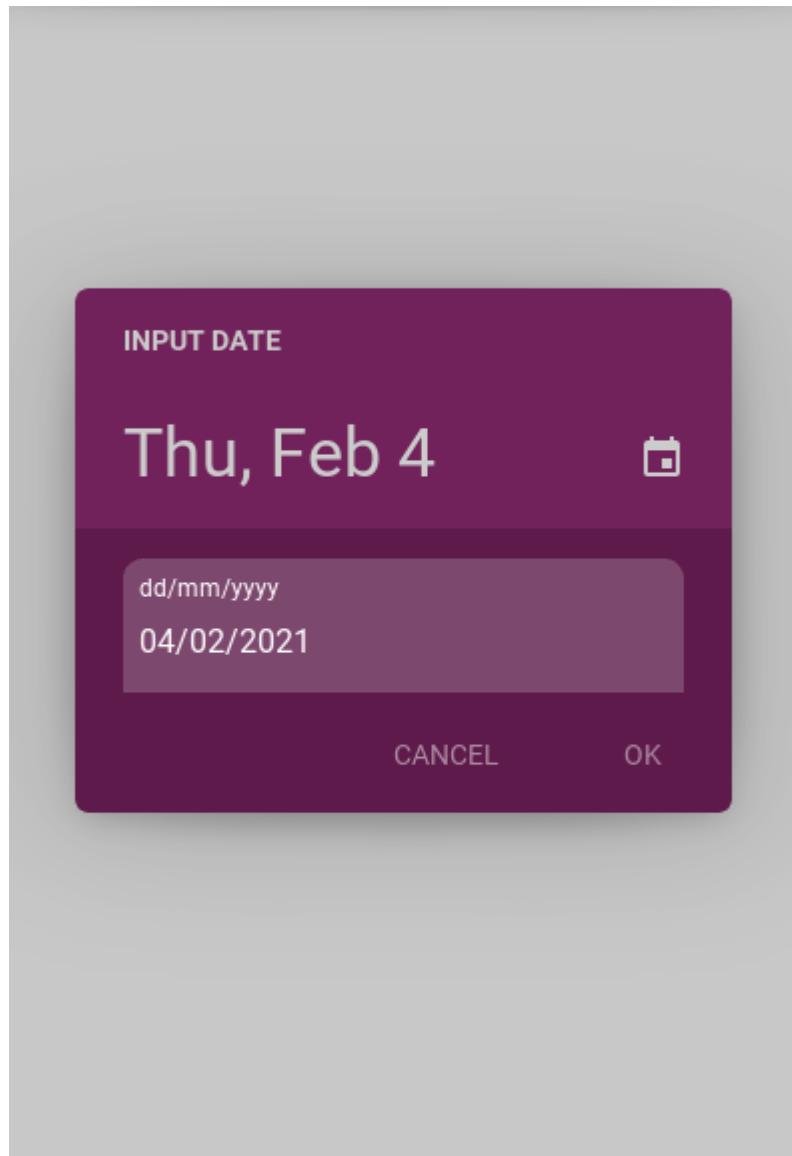
`input_field_background_color` is an `ColorProperty` and defaults to `None`.

#### `input_field_text_color`

Text color of input fields.

Background color of input fields.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffff",
 text_current_color=get_color_from_hex("#e93f39"),
 input_field_background_color=(1, 1, 1, 0.2),
 input_field_text_color=(1, 1, 1, 1),
)
```

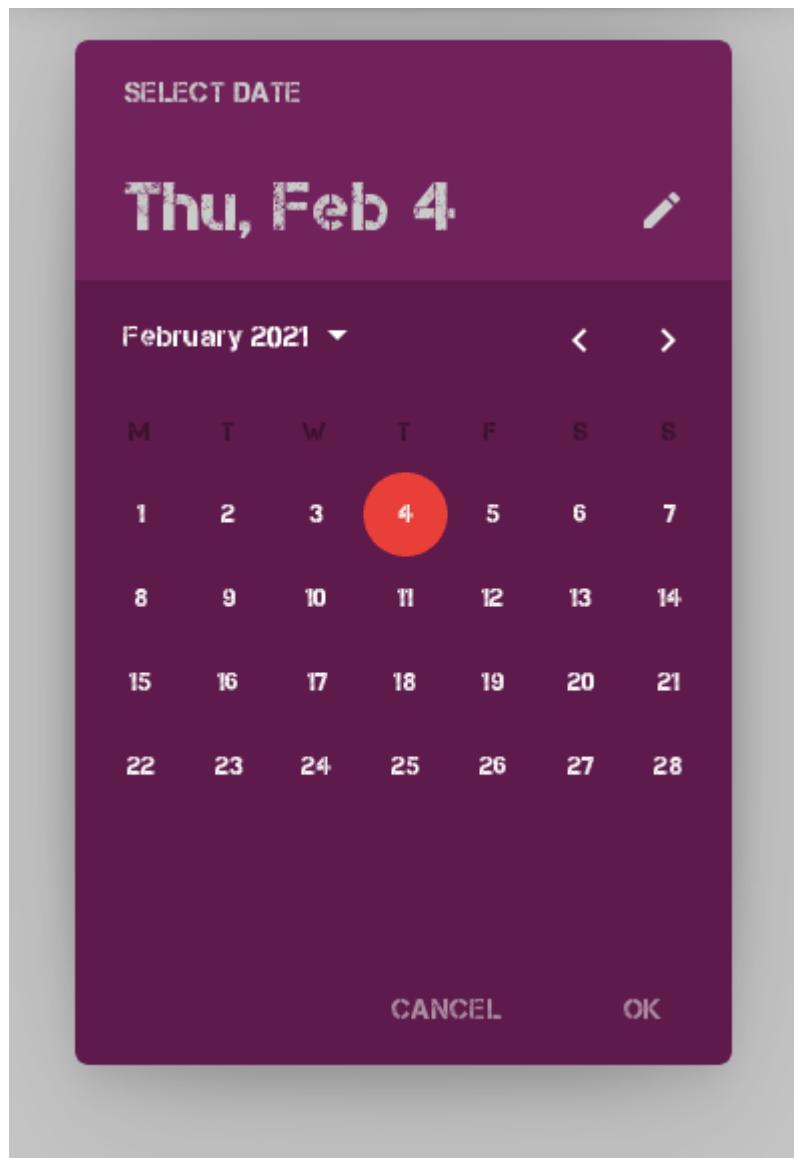


`input_field_text_color` is an `ColorProperty` and defaults to `None`.

**font\_name**

Font name for dialog window text.

```
MDDatePicker(
 primary_color=get_color_from_hex("#72225b"),
 accent_color=get_color_from_hex("#5d1a4a"),
 selector_color=get_color_from_hex("#e93f39"),
 text_toolbar_color=get_color_from_hex("#cccccc"),
 text_color="#ffffffff",
 text_current_color=get_color_from_hex("#e93f39"),
 input_field_background_color=(1, 1, 1, 0.2),
 input_field_text_color=(1, 1, 1, 1),
 font_name="Weather.ttf",
)
```



*font\_name* is an `StringProperty` and defaults to ‘*Roboto*’.

#### `on_save(self, *args)`

Events called when the “OK” dialog box button is clicked.

#### `on_cancel(self, *args)`

Events called when the “CANCEL” dialog box button is clicked.

**class** `kivymd.uix.picker.MDDatePicker(year=None, month=None, day=None, firstweekday=0, **kwargs)`  
Base class for `MDDatePicker` and `MDTIMEPicker` classes.

#### Events

`on_save` Events called when the “OK” dialog box button is clicked.

`on_cancel` Events called when the “CANCEL” dialog box button is clicked.

#### `text_weekday_color`

Text color of weekday names.

`text_weekday_color` is an `ColorProperty` and defaults to *None*.

**day**

The day of the month to be opened by default. If not specified, the current number will be used.

`day` is an [NumericProperty](#) and defaults to `0`.

**month**

The number of month to be opened by default. If not specified, the current number will be used.

`month` is an [NumericProperty](#) and defaults to `0`.

**year**

The year of month to be opened by default. If not specified, the current number will be used.

`year` is an [NumericProperty](#) and defaults to `0`.

**min\_year**

The year of month to be opened by default. If not specified, the current number will be used.

`min_year` is an [NumericProperty](#) and defaults to `1914`.

**max\_year**

The year of month to be opened by default. If not specified, the current number will be used.

`max_year` is an [NumericProperty](#) and defaults to `2121`.

**mode**

**Dialog type:** ‘`picker`’ type allows you to select one date; ‘`range`’ type allows to set a range of dates from which the user can select a date.

Available options are: `['picker', 'range']`.

`mode` is an [OptionProperty](#) and defaults to `picker`.

**min\_date**

The minimum value of the date range for the ‘`mode`’ parameter. Must be an object <class ‘`datetime.date`’>.

`min_date` is an [ObjectProperty](#) and defaults to `None`.

**max\_date**

The minimum value of the date range for the ‘`mode`’ parameter. Must be an object <class ‘`datetime.date`’>.

`max_date` is an [ObjectProperty](#) and defaults to `None`.

**date\_range\_text\_error**

Error text that will be shown on the screen in the form of a toast if the minimum date range exceeds the maximum.

`date_range_text_error` is an [StringProperty](#) and defaults to ‘`Error date range`’.

**sel\_year**

**sel\_month**

**sel\_day**

**on\_device\_orientation(self, instance, value)**

**transformation\_from\_dialog\_select\_year(self)**

**transformation\_to\_dialog\_select\_year(self)**

**transformation\_to\_dialog\_input\_date(self)**

**transformation\_from\_dialog\_input\_date(self, interval)**

**compare\_date\_range(self)**

---

**update\_calendar\_for\_date\_range(self)**

**update\_text\_full\_date(self, list\_date)**  
Updates the title of the week, month and number day name in an open date input dialog.

**update\_calendar(self, year, month)**

**get\_field(self)**  
Creates and returns a text field object used to enter dates.

**get\_date\_range(self)**

**set\_text\_full\_date(self, year, month, day, orientation)**  
Returns a string of type “Tue, Feb 2” or “Tue, Feb 2” for a date  
choose and a string like “Feb 15 - Mar 23” or “Feb 15,  
**Mar 23” for a date range.**

**set\_selected\_widget(self, widget)**

**set\_month\_day(self, day)**

**set\_position\_to\_current\_year(self)**

**generate\_list\_widgets\_years(self)**

**generate\_list\_widgets\_days(self)**

**change\_month(self, operation)**  
Called when “chevron-left” and “chevron-right” buttons are pressed. Switches the calendar to the previous/next month.

**class kivymd.uix.picker.MDTimePicker(\*\*kwargs)**  
Base class for [MDDatePicker](#) and [MDTimePicker](#) classes.

**Events**

**on\_save** Events called when the “OK” dialog box button is clicked.

**on\_cancel** Events called when the “CANCEL” dialog box button is clicked.

**hour**  
Current hour  
*hour* is an [StringProperty](#) and defaults to ‘12’.

**minute**  
Current minute  
*minute* is an [StringProperty](#) and defaults to 0.

**minute\_radius**  
Radius of the minute input field.  
*minute\_radius* is an [ListProperty](#) and defaults to [dp(5),].

**hour\_radius**  
Radius of the hour input field.  
*hour\_radius* is an [ListProperty](#) and defaults to [dp(5),].

**am\_pm\_radius**  
Radius of the AM/PM selector.  
*am\_pm\_radius* is an [NumericProperty](#) and defaults to dp(5).

### `am_pm_border_width`

Width of the AM/PM selector's borders.

`am_pm_border_width` is an `NumericProperty` and defaults to `dp(1)`.

### `am_pm`

Current AM/PM mode.

`am_pm` is an `OptionProperty` and defaults to ‘`am`’.

### `animation_duration`

Duration of the animations.

`animation_duration` is an `NumericProperty` and defaults to `0.2`.

### `animation_transition`

Transition type of the animations.

`animation_transition` is an `StringProperty` and defaults to ‘`out_quad`’.

### `time`

Returns the current time object.

`time` is an `ObjectProperty` and defaults to `None`.

### `set_time(self, time_obj)`

Manually set time dialog with the specified time.

### `get_state(self)`

Returns the current state of TimePicker. Can be one of `portrait`, `landscape` or `input`.

## `class kivymd.uix.picker.MDThemePicker(**kwargs)`

ModalView class. See module documentation for more information.

### Events

`on_pre_open`: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

`on_open`: Fired when the ModalView is opened.

`on_pre_dismiss`: Fired before the ModalView is closed.

`on_dismiss`: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events `on_pre_open` and `on_pre_dismiss`.

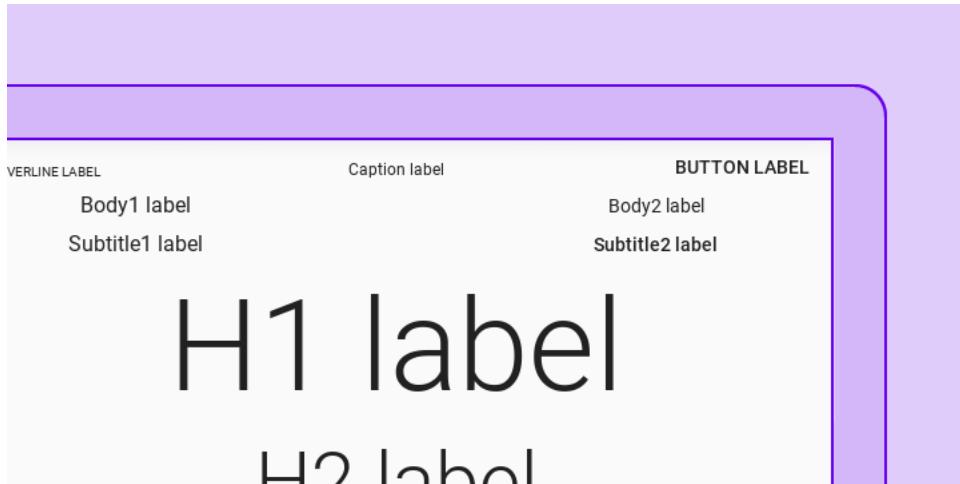
Changed in version 2.0.0: Added property ‘`overlay_color`’.

### `on_open(self)`

### `on_tab_switch(self, instance_tabs, instance_tab, instance_tab_label, tab_text)`

### 2.3.21 Label

The `MDLabel` widget is for rendering text.



- `MDLabel`
- `MDIcon`

#### MDLabel

Class `MDLabel` inherited from the `Label` class but for `MDLabel` the `text_size` parameter is `(self.width, None)` and default is positioned on the left:

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = """
MDScreen:

 MDBBoxLayout:
 orientation: "vertical"

 MDToolbar:
 title: "MDLabel"

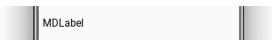
 MDLabel:
 text: "MDLabel"
 ...

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)
```

(continues on next page)

(continued from previous page)

Test().run()

**Note:** See `halign` and `valign` attributes of the `Label` class**MDLabel:**

```
text: "MDLabel"
halign: "center"
```

**MDLabel color:**`MDLabel` provides standard color themes for label color management:

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.label import MDLabel

KV = '''
MDScreen:

 MDBBoxLayout:
 id: box
 orientation: "vertical"

 MDToolbar:
 title: "MDLabel"
 '''

class Test(MDApp):
 def build(self):
 screen = Builder.load_string(KV)
 # Names of standard color themes.
 for name_theme in [
 "Primary",
 "Secondary",
 "Hint",
 "Error",
 "ContrastParentBackground",
]:
 screen.ids.box.add_widget(
 MDLabel(
 text=name_theme,
 halign="center",

```

(continues on next page)

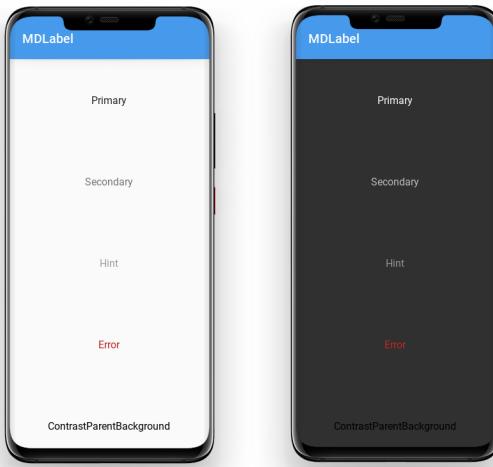
(continued from previous page)

```

 theme_text_color=name_theme,
)
)
return screen

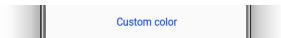
```

```
Test().run()
```



To use a custom color for `MDLabel`, use a theme ‘Custom’. After that, you can specify the desired color in the `rgba` format in the `text_color` parameter:

```
MDLabel:
 text: "Custom color"
 halign: "center"
 theme_text_color: "Custom"
 text_color: 0, 0, 1, 1
```



`MDLabel` provides standard font styles for labels. To do this, specify the name of the desired style in the `font_style` parameter:

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.label import MDLabel
from kivymd.font_definitions import theme_font_styles

KV = '''
MDScreen:

 MDBBoxLayout:
 orientation: "vertical"

 MToolbar:
```

(continues on next page)

(continued from previous page)

```
 title: "MDLabel"

 ScrollView:

 MDList:
 id: box
 ...

class Test(MDApp):
 def build(self):
 screen = Builder.load_string(KV)
 # Names of standard font styles.
 for name_style in theme_font_styles[:-1]:
 screen.ids.box.add_widget(
 MDLabel(
 text=f'{name_style} style',
 halign="center",
 font_style=name_style,
)
)
 return screen

Test().run()
```

## MDIcon

You can use labels to display material design icons using the [MDIcon](#) class.

### See also:

[Material Design Icons](#)

[Material Design Icon Names](#)

The [MDIcon](#) class is inherited from [MDLabel](#) and has the same parameters.

**Warning:** For the [MDIcon](#) class, you cannot use `text` and `font_style` options!

```
MDIcon:
 halign: "center"
 icon: "language-python"
```



**API - kivymd.uix.label****class** kivymd.uix.label.**MDLabel**(\*\*kwargs)

Label class, see module documentation for more information.

**Events***on\_ref\_press* Fired when the user clicks on a word referenced with a [ref] tag in a text markup.**font\_style**

Label font style.

Available vanilla font\_style are: ‘H1’, ‘H2’, ‘H3’, ‘H4’, ‘H5’, ‘H6’, ‘Subtitle1’, ‘Subtitle2’, ‘Body1’, ‘Body2’, ‘Button’, ‘Caption’, ‘Overline’, ‘Icon’.

*font\_style* is an **StringProperty** and defaults to ‘Body1’.**text**

Text of the label.

**theme\_text\_color**

Label color scheme name.

Available options are: ‘Primary’, ‘Secondary’, ‘Hint’, ‘Error’, ‘Custom’, ‘ContrastParentBackground’.

*theme\_text\_color* is an **OptionProperty** and defaults to *None*.**text\_color**Label text color in **rgba** format.*text\_color* is an **ColorProperty** and defaults to *None*.**parent\_background****can\_capitalize****check\_font\_styles(self, \*dt)****update\_font\_style(self, \*args)****on\_theme\_text\_color(self, instance, value)****on\_text\_color(self, \*args)****on\_opposite\_colors(self, instance, value)****class** kivymd.uix.label.**MDIcon**(\*\*kwargs)

Label class, see module documentation for more information.

**Events***on\_ref\_press* Fired when the user clicks on a word referenced with a [ref] tag in a text markup.**icon**

Label icon name.

*icon* is an **StringProperty** and defaults to ‘android’.**source**

Path to icon.

*source* is an **StringProperty** and defaults to *None*.

## 2.3.22 TapTargetView

See also:

[TapTargetView](#), [GitHub](#)

[TapTargetView](#), [Material archive](#)

Provide value and improve engagement by introducing users to new features and functionality at relevant moments.

### Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.taptargetview import MDTapTargetView

KV = '''
Screen:

 MDFloatingActionButton:
 id: button
 icon: "plus"
 pos: 10, 10
 on_release: app.tap_target_start()
'''

class TapTargetViewDemo(MDApp):
 def build(self):
 screen = Builder.load_string(KV)
 self.tap_target_view = MDTapTargetView(
 widget=screen.ids.button,
 title_text="This is an add button",
 description_text="This is a description of the button",
 widget_position="left_bottom",
)

 return screen

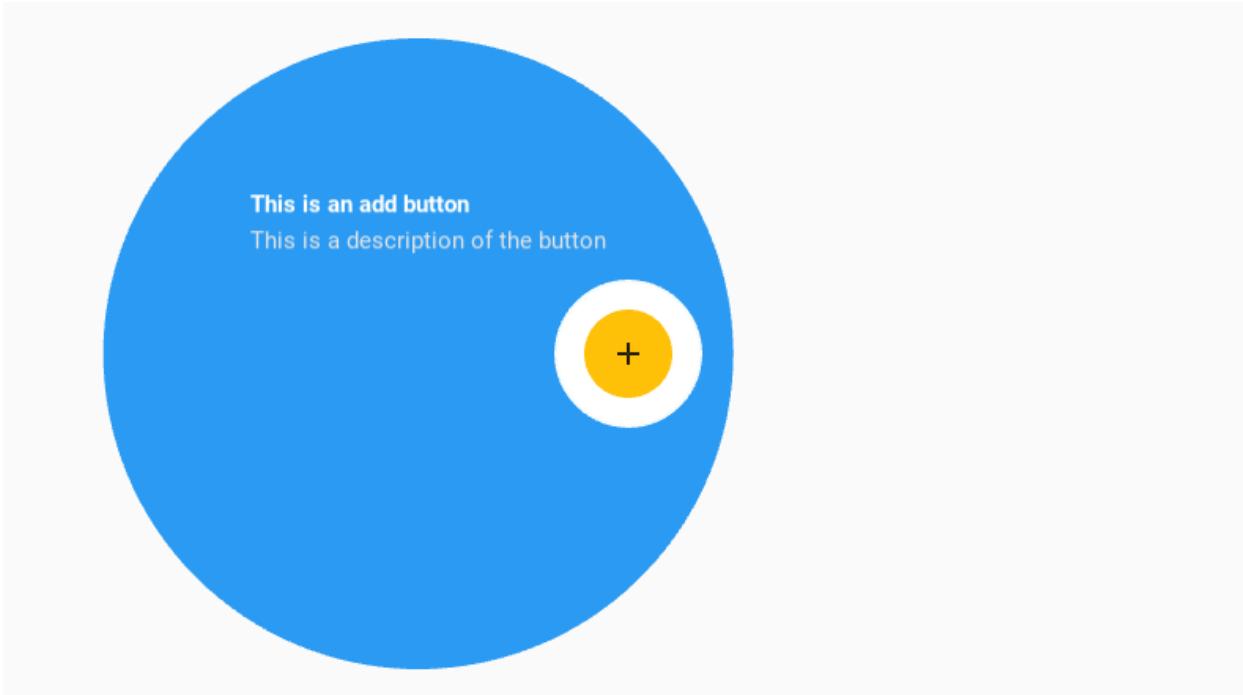
 def tap_target_start(self):
 if self.tap_target_view.state == "close":
 self.tap_target_view.start()
 else:
 self.tap_target_view.stop()

TapTargetViewDemo().run()
```

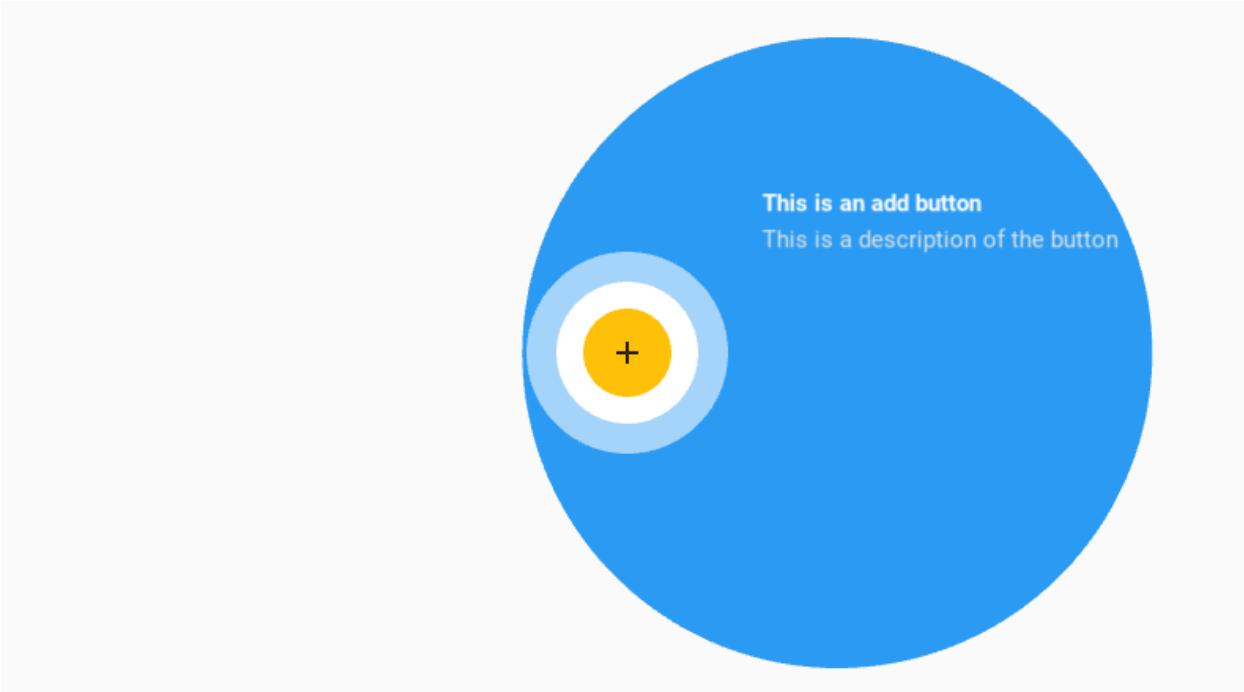
## Widget position

Sets the position of the widget relative to the floating circle.

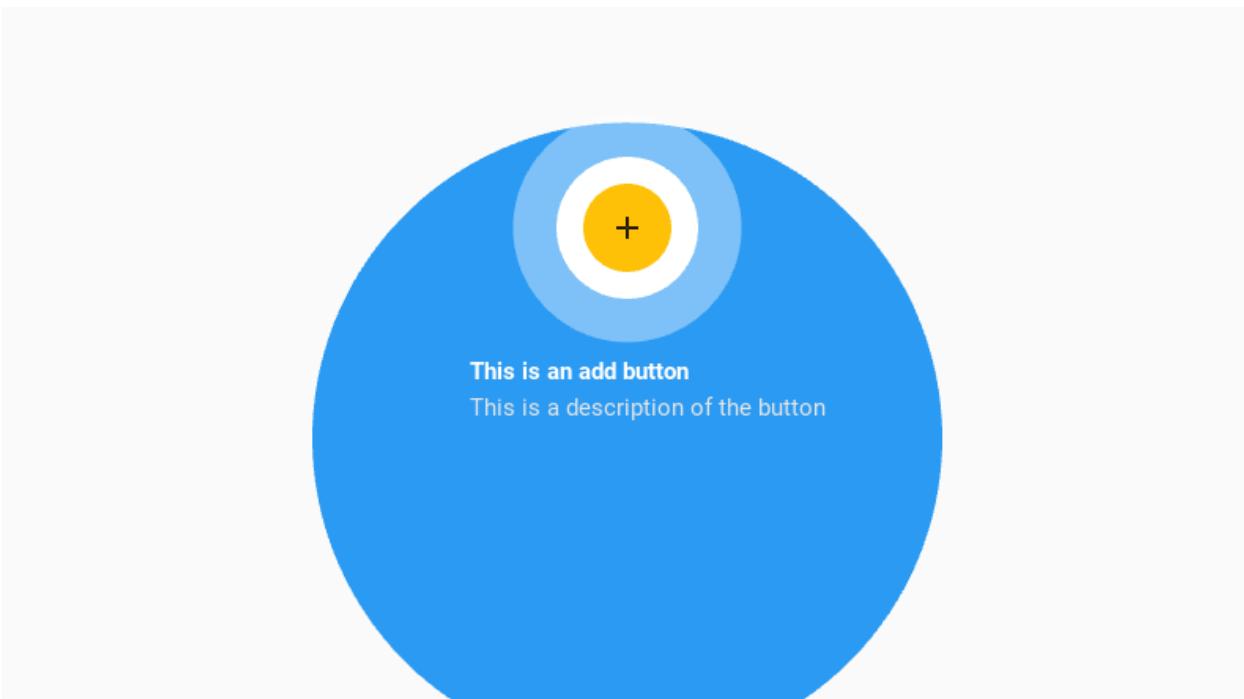
```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="right",
)
```



```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="left",
)
```



```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="top",
)
```

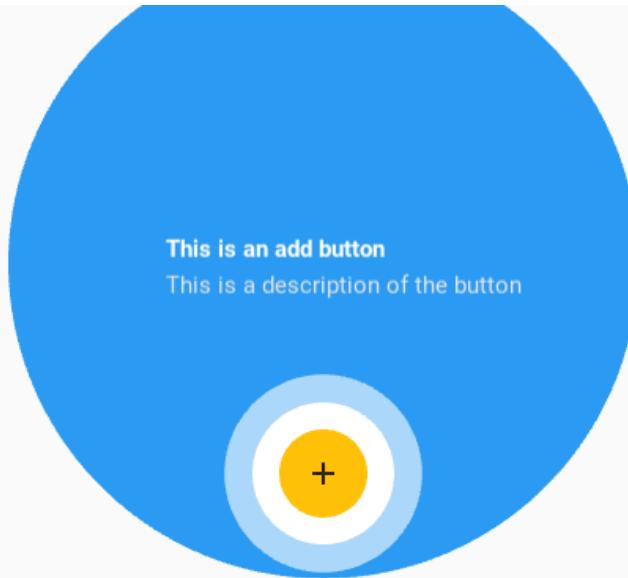


```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="bottom",
```

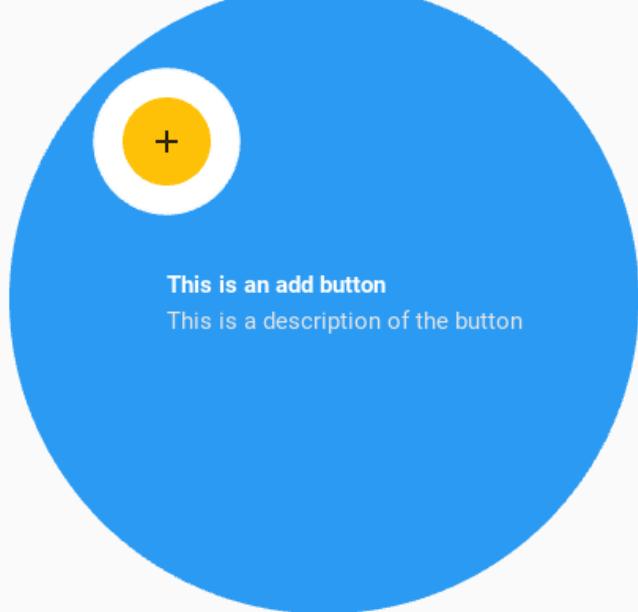
(continues on next page)

(continued from previous page)

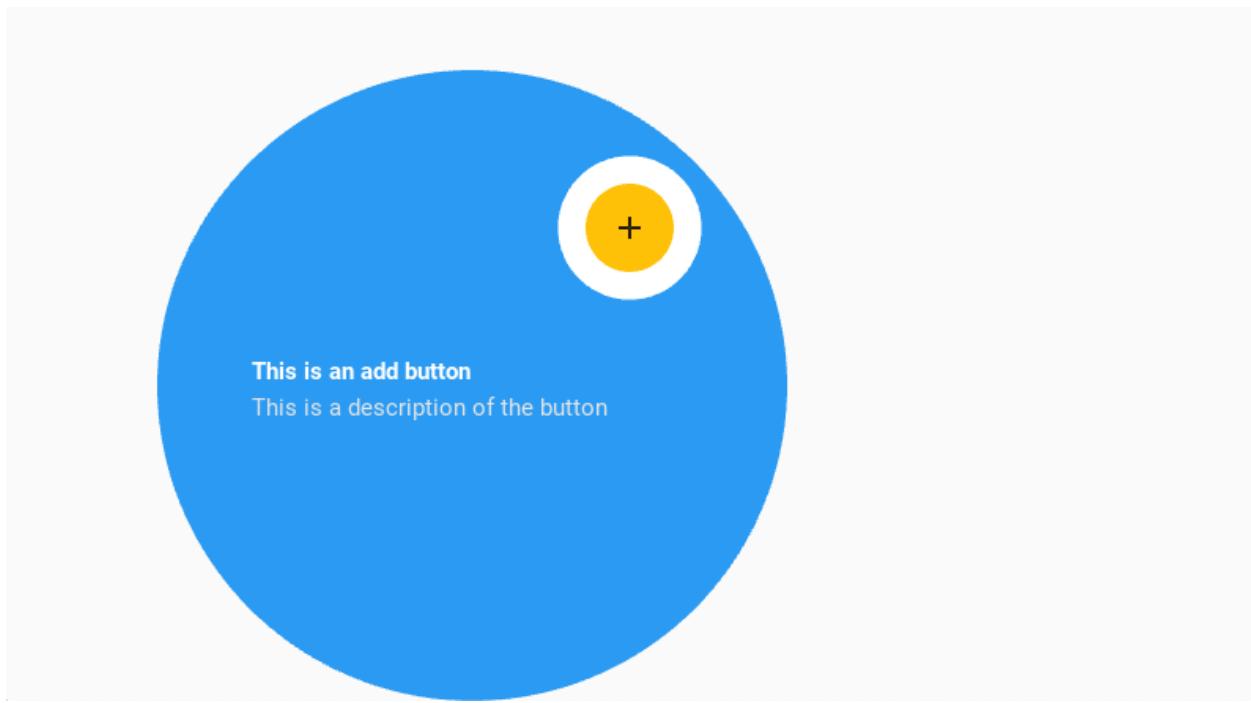
)



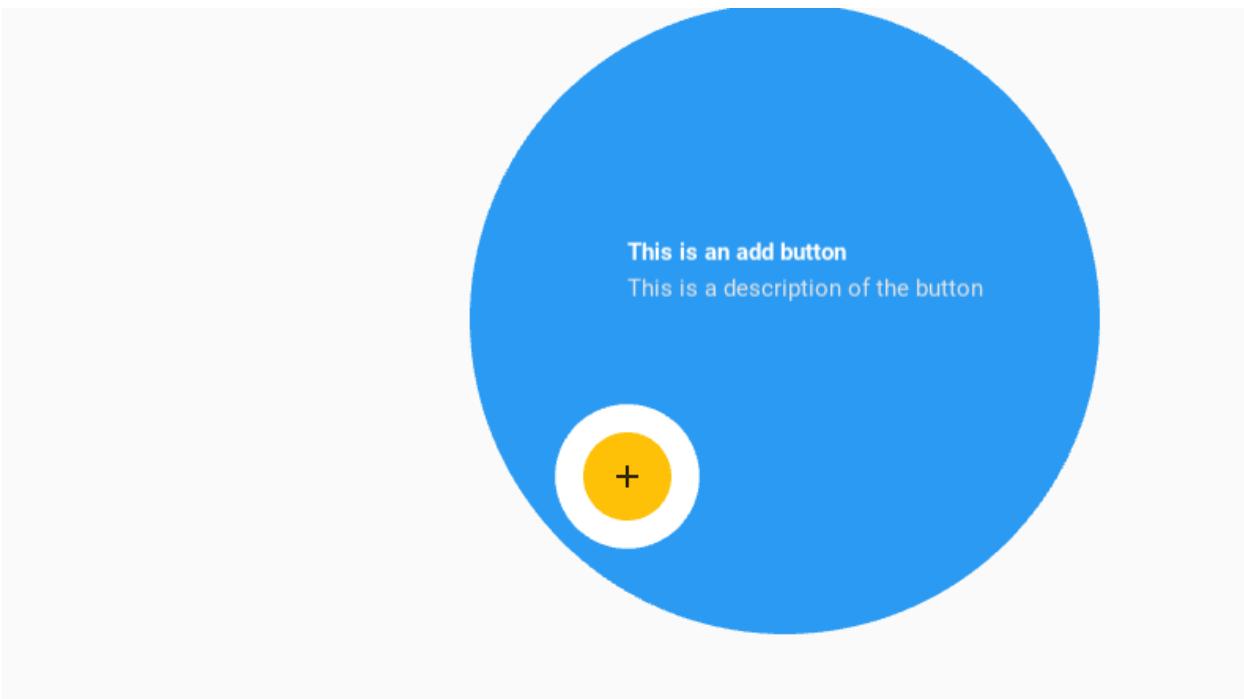
```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="left_top",
)
```



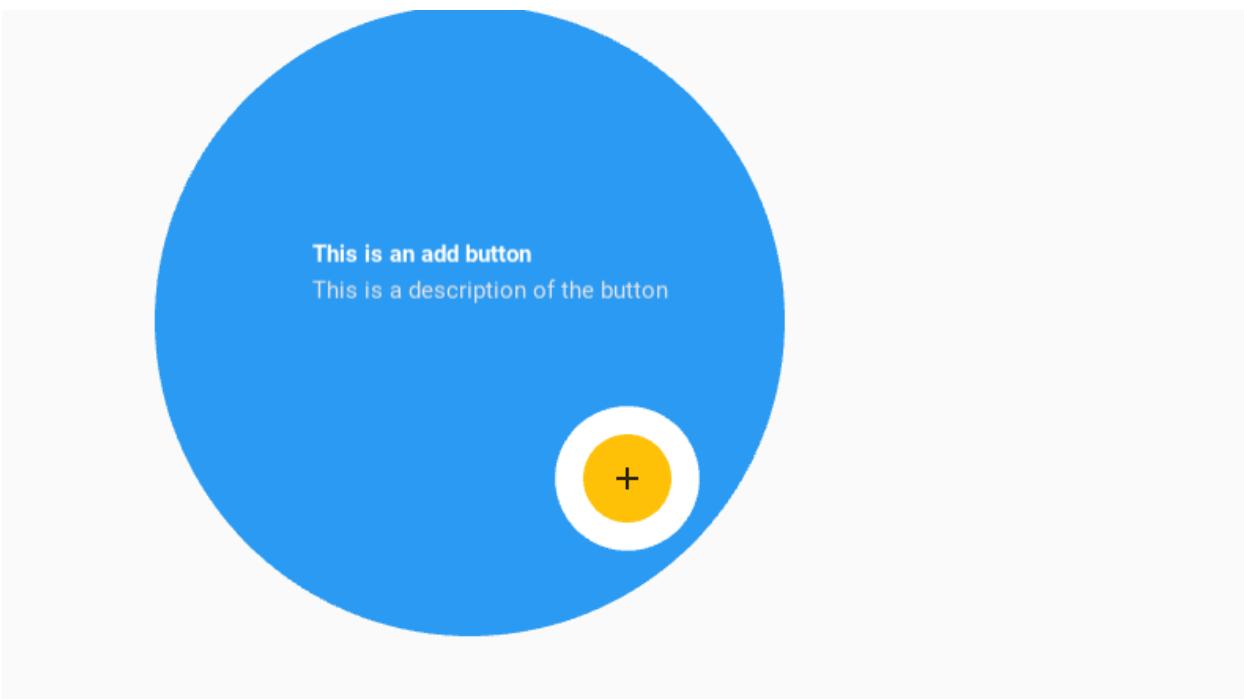
```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="right_top",
)
```



```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="left_bottom",
)
```



```
self.tap_target_view = MDTapTargetView(
 ...
 widget_position="right_bottom",
)
```



If you use the `widget_position = "center"` parameter then you must definitely specify the `title_position`.

```
self.tap_target_view = MDTapTargetView(
```

(continues on next page)

(continued from previous page)

```
...
 widget_position="center",
 title_position="left_top",
)
```



### Text options

```
self.tap_target_view = MDTapTargetView(
 ...
 title_text="Title text",
 description_text="Description text",
)
```



You can use the following options to control font size, color, and boldness:

- `title_text_size`
- `title_text_color`
- `title_text_bold`
- `description_text_size`
- `description_text_color`
- `description_text_bold`

```
self.tap_target_view = MDTapTargetView(
 ...
 title_text="Title text",
 title_text_size="36sp",
 description_text="Description text",
 description_text_color=[1, 0, 0, 1]
)
```



But you can also use markup to set these values.

```
self.tap_target_view = MDTapTargetView(
 ...
 title_text="[size=36]Title text[/size]",
 description_text="[color=#ff0000ff]Description text[/color]",
)
```

### Events control

```
self.tap_target_view.bind(on_open=self.on_open, on_close=self.on_close)
```

```
def on_open(self, instance_tap_target_view):
 "Called at the time of the start of the widget opening animation."

 print("Open", instance_tap_target_view)

def on_close(self, instance_tap_target_view):
 "Called at the time of the start of the widget closed animation."

 print("Close", instance_tap_target_view)
```

---

**Note:** See other parameters in the `MDTapTargetView` class.

---

**API - kivymd.uix.taptargetview**

```
class kivymd.uix.taptargetview.MDTapTargetView(**kwargs)
```

Rough try to mimic the working of Android's TapTargetView.

**Events**

**on\_open** Called at the time of the start of the widget opening animation.

**on\_close** Called at the time of the start of the widget closed animation.

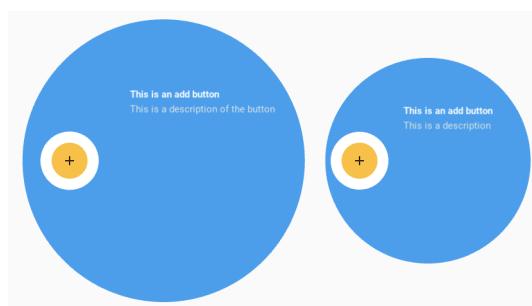
**widget**

Widget to add TapTargetView upon.

*widget* is an `ObjectProperty` and defaults to `None`.

**outer\_radius**

Radius for outer circle.



*outer\_radius* is an `NumericProperty` and defaults to `dp(200)`.

**outer\_circle\_color**

Color for the outer circle in `rgb` format.

```
self.tap_target_view = MDTapTargetView(
 ...
 outer_circle_color=(1, 0, 0)
)
```



`outer_circle_color` is an `ListProperty` and defaults to `theme_cls.primary_color`.

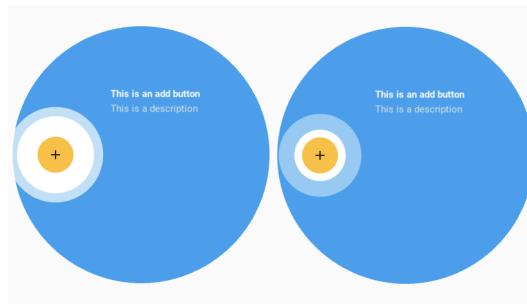
**outer\_circle\_alpha**

Alpha value for outer circle.

`outer_circle_alpha` is an `NumericProperty` and defaults to `0.96`.

**target\_radius**

Radius for target circle.

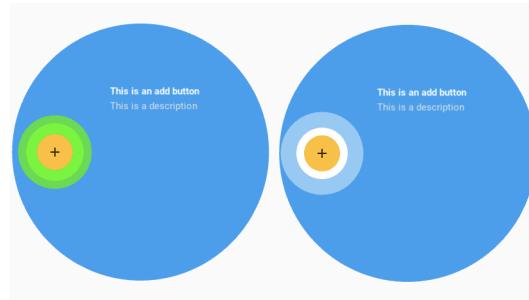


`target_radius` is an `NumericProperty` and defaults to `dp(45)`.

**target\_circle\_color**

Color for target circle in `rgb` format.

```
self.tap_target_view = MDTapTargetView(
 ...
 target_circle_color=(1, 0, 0)
)
```



`target_circle_color` is an `ListProperty` and defaults to `[1, 0, 0]`.

#### **title\_text**

Title to be shown on the view.

`title_text` is an `StringProperty` and defaults to “”.

#### **title\_text\_size**

Text size for title.

`title_text_size` is an `NumericProperty` and defaults to `dp(25)`.

#### **title\_text\_color**

Text color for title.

`title_text_color` is an `ListProperty` and defaults to `[1, 1, 1, 1]`.

#### **title\_text\_bold**

Whether title should be bold.

`title_text_bold` is an `BooleanProperty` and defaults to `True`.

#### **description\_text**

Description to be shown below the title (keep it short).

`description_text` is an `StringProperty` and defaults to “”.

#### **description\_text\_size**

Text size for description text.

`description_text_size` is an `NumericProperty` and defaults to `dp(20)`.

#### **description\_text\_color**

Text size for description text.

`description_text_color` is an `ListProperty` and defaults to `[0.9, 0.9, 0.9, 1]`.

#### **description\_text\_bold**

Whether description should be bold.

`description_text_bold` is an `BooleanProperty` and defaults to `False`.

#### **draw\_shadow**

Whether to show shadow.

`draw_shadow` is an `BooleanProperty` and defaults to `False`.

**cancelable**

Whether clicking outside the outer circle dismisses the view.

`cancelable` is an `BooleanProperty` and defaults to `False`.

**widget\_position**

Sets the position of the widget on the `outer_circle`. Available options are `'left'`, `'right'`, `'top'`, `'bottom'`, `'left_top'`, `'right_top'`, `'left_bottom'`, `'right_bottom'`, `'center'`.

`widget_position` is an `OptionProperty` and defaults to `'left'`.

**title\_position**

Sets the position of `:attr`~title_text`` on the outer circle. Only works if `:attr`~widget_position`` is set to `'center'`. In all other cases, it calculates the `:attr`~title_position`` itself. Must be set to other than `'auto'` when `:attr`~widget_position`` is set to `'center'`.

Available options are `'auto'`, `'left'`, `'right'`, `'top'`, `'bottom'`, `'left_top'`, `'right_top'`, `'left_bottom'`, `'right_bottom'`, `'center'`.

`title_position` is an `OptionProperty` and defaults to `'auto'`.

**stop\_on\_outer\_touch**

Whether clicking on outer circle stops the animation.

`stop_on_outer_touch` is an `BooleanProperty` and defaults to `False`.

**stop\_on\_target\_touch**

Whether clicking on target circle should stop the animation.

`stop_on_target_touch` is an `BooleanProperty` and defaults to `True`.

**state**

State of `MDTapTargetView`.

`state` is an `OptionProperty` and defaults to `'close'`.

**stop(self, \*args)**

Starts widget close animation.

**start(self, \*args)**

Starts widget opening animation.

**on\_open(self, \*args)**

Called at the time of the start of the widget opening animation.

**on\_close(self, \*args)**

Called at the time of the start of the widget closed animation.

**on\_draw\_shadow(self, instance, value)****on\_description\_text(self, instance, value)****on\_description\_text\_size(self, instance, value)****on\_description\_text\_bold(self, instance, value)****on\_title\_text(self, instance, value)****on\_title\_text\_size(self, instance, value)****on\_title\_text\_bold(self, instance, value)****on\_outer\_radius(self, instance, value)****on\_target\_radius(self, instance, value)****on\_target\_touch(self)**

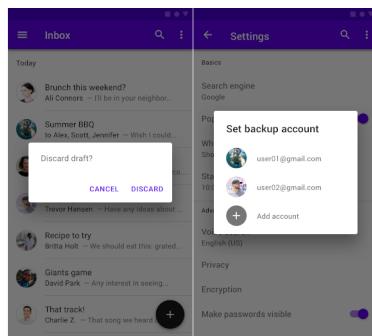
```
on_outer_touch(self)
on_outside_click(self)
```

### 2.3.23 Dialog

#### See also:

Material Design spec, Dialogs

Dialogs inform users about a task and can contain critical information, require decisions, or involve multiple tasks.



#### Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.button import MDFloatButton
from kivymd.uix.dialog import MDDialog

KV = '''
MDFloatLayout:

 MDFloatButton:
 text: "ALERT DIALOG"
 pos_hint: {'center_x': .5, 'center_y': .5}
 on_release: app.show_alert_dialog()
'''

class Example(MDApp):
 dialog = None

 def build(self):
 return Builder.load_string(KV)

 def show_alert_dialog(self):
```

(continues on next page)

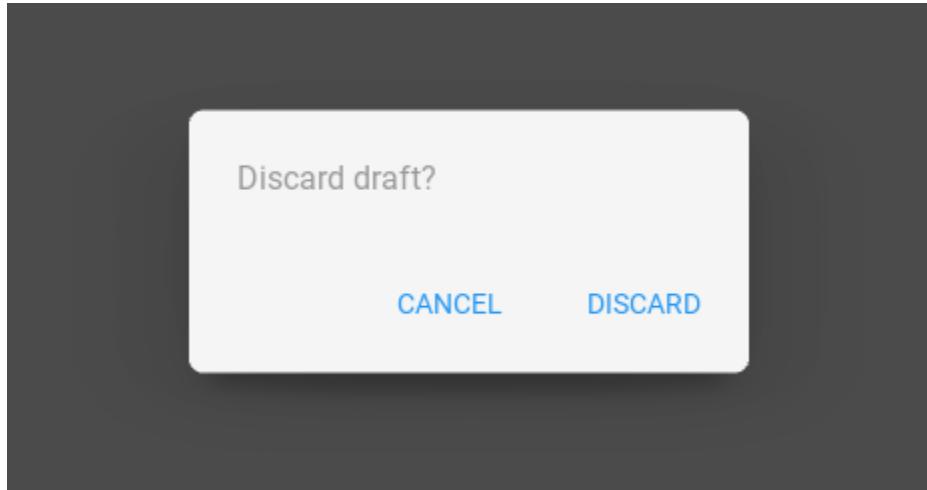
(continued from previous page)

```

if not self.dialog:
 self.dialog = MDDialog(
 text="Discard draft?",
 buttons=[
 MDFlatButton(
 text="CANCEL", text_color=self.theme_cls.primary_color
),
 MDFlatButton(
 text="DISCARD", text_color=self.theme_cls.primary_color
),
],
)
self.dialog.open()

```

Example().run()



## API - kivymd.uix.dialog

**class kivymd.uix.dialog.MDDialog(\*\*kwargs)**

ModalView class. See module documentation for more information.

### Events

***on\_pre\_open***: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

***on\_open***: Fired when the ModalView is opened.

***on\_pre\_dismiss***: Fired before the ModalView is closed.

***on\_dismiss***: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

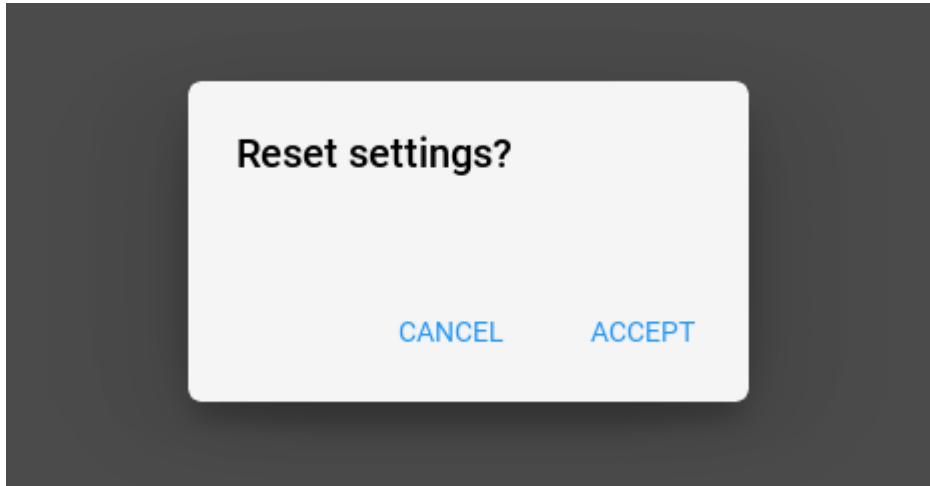
Changed in version 1.11.0: Added events *on\_pre\_open* and *on\_pre\_dismiss*.

Changed in version 2.0.0: Added property ‘overlay\_color’.

### title

Title dialog.

```
[...]
 self.dialog = MDDialog(
 title="Reset settings?",
 buttons=[
 MDFlatButton(
 text="CANCEL", text_color=self.theme_cls.primary_color
),
 MDFlatButton(
 text="ACCEPT", text_color=self.theme_cls.primary_color
),
],
)
[...]
```

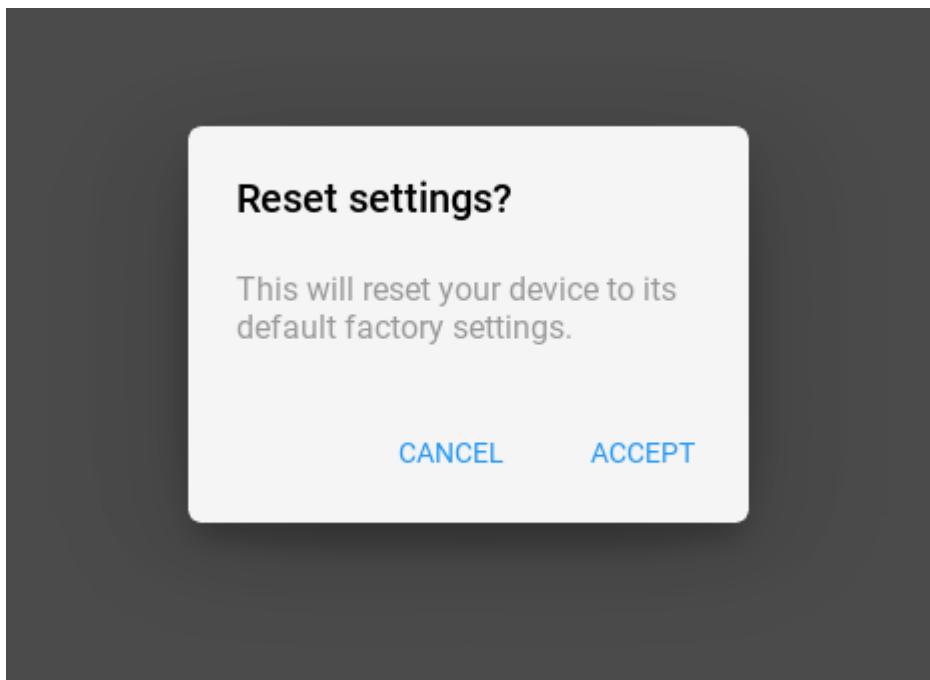


`title` is an `StringProperty` and defaults to ''.

#### text

Text dialog.

```
[...]
 self.dialog = MDDialog(
 title="Reset settings?",
 text="This will reset your device to its default factory settings.",
 buttons=[
 MDFlatButton(
 text="CANCEL", text_color=self.theme_cls.primary_color
),
 MDFlatButton(
 text="ACCEPT", text_color=self.theme_cls.primary_color
),
],
)
[...]
```

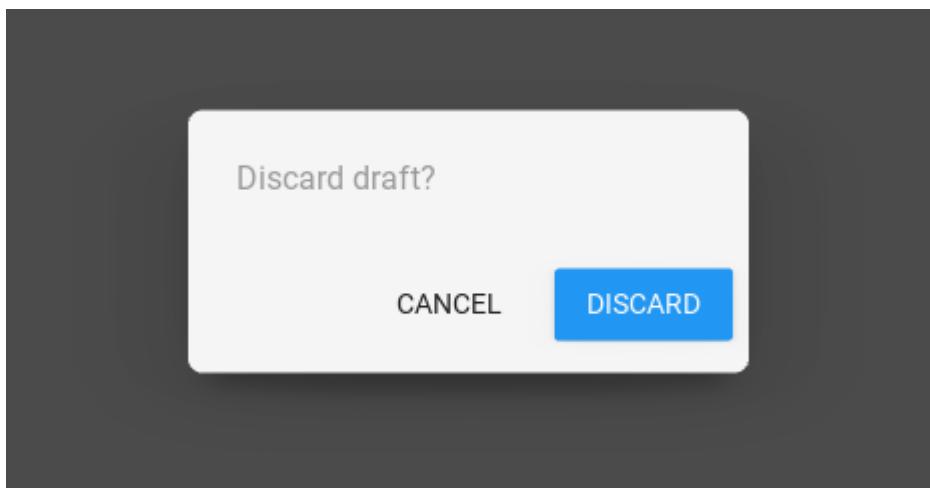


`text` is an `StringProperty` and defaults to ''.

#### buttons

List of button objects for dialog. Objects must be inherited from `BaseButton` class.

```
[...]
 self.dialog = MDDialog(
 text="Discard draft?",
 buttons=[
 MDFlatButton(text="CANCEL"),
 MDRaisedButton(text="DISCARD"),
],
)
[...]
```



`buttons` is an `ListProperty` and defaults to [].

#### items

List of items objects for dialog. Objects must be inherited from `BaseListItem` class.

```

from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.dialog import MDDialog
from kivymd.uix.list import OneLineAvatarListItem

KV = '''
<Item>

 ImageLeftWidget:
 source: root.source

MDFloatLayout:

 MDFlatButton:
 text: "ALERT DIALOG"
 pos_hint: {'center_x': .5, 'center_y': .5}
 on_release: app.show_simple_dialog()
'''

class Item(OneLineAvatarListItem):
 divider = None
 source = StringProperty()

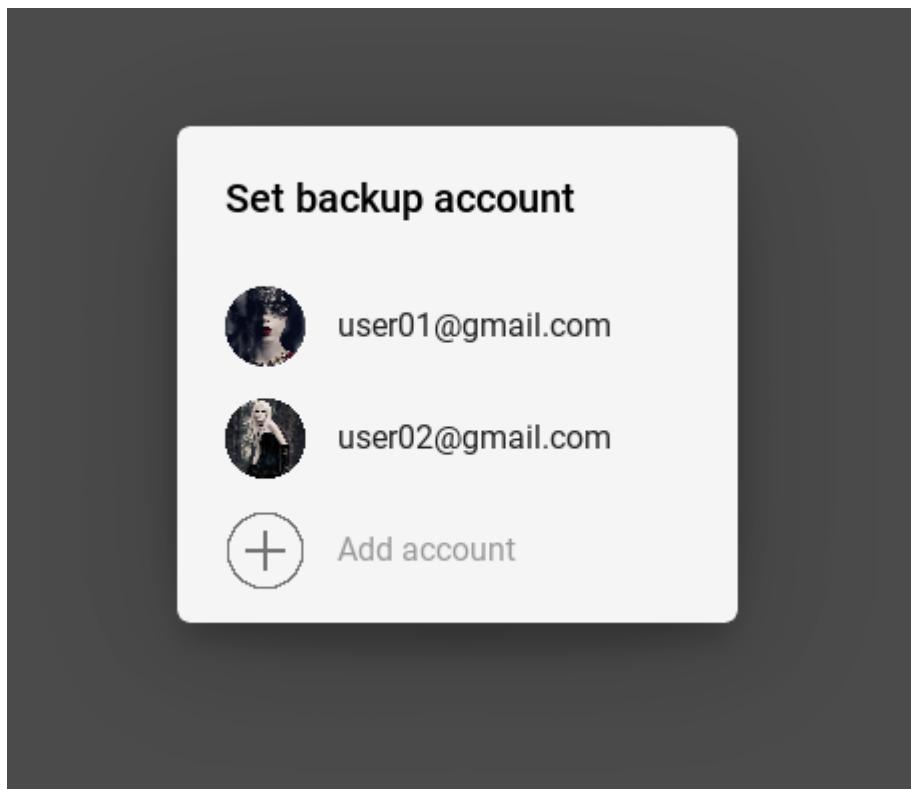
class Example(MDApp):
 dialog = None

 def build(self):
 return Builder.load_string(KV)

 def show_simple_dialog(self):
 if not self.dialog:
 self.dialog = MDDialog(
 title="Set backup account",
 type="simple",
 items=[
 Item(text="user01@gmail.com", source="user-1.png"),
 Item(text="user02@gmail.com", source="user-2.png"),
 Item(text="Add account", source="add-icon.png"),
],
)
 self.dialog.open()

Example().run()

```



```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.dialog import MDDialog
from kivymd.uix.list import OneLineAvatarIconListItem

KV = '''
<ItemConfirm>
 on_release: root.set_icon(check)

 CheckboxLeftWidget:
 id: check
 group: "check"

MDFloatLayout:

 MDFlatButton:
 text: "ALERT DIALOG"
 pos_hint: {'center_x': .5, 'center_y': .5}
 on_release: app.show_confirmation_dialog()
'''

class ItemConfirm(OneLineAvatarIconListItem):
 divider = None
```

(continues on next page)

(continued from previous page)

```

def set_icon(self, instance_check):
 instance_check.active = True
 check_list = instance_check.get_widgets(instance_check.group)
 for check in check_list:
 if check != instance_check:
 check.active = False

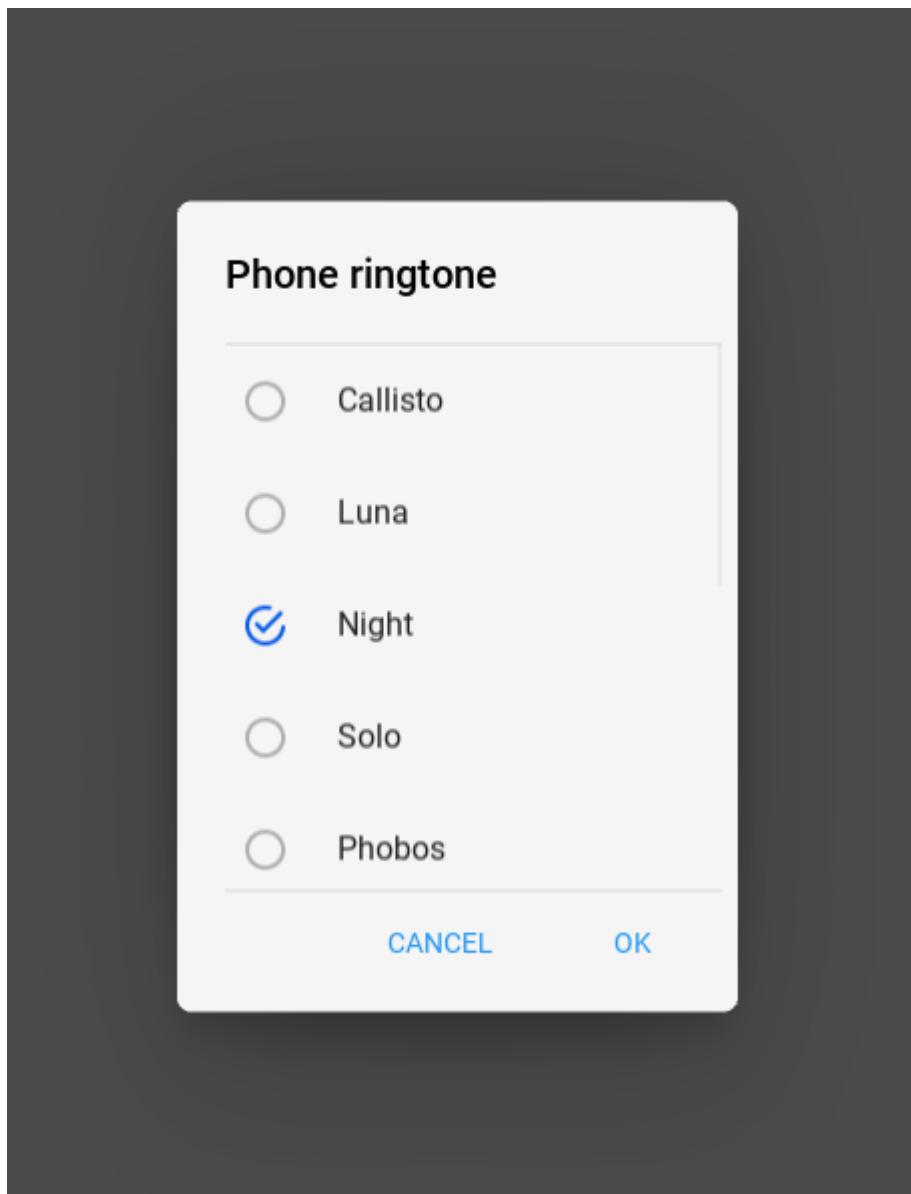
class Example(MDApp):
 dialog = None

 def build(self):
 return Builder.load_string(KV)

 def show_confirmation_dialog(self):
 if not self.dialog:
 self.dialog = MDDialog(
 title="Phone ringtone",
 type="confirmation",
 items=[
 ItemConfirm(text="Callisto"),
 ItemConfirm(text="Luna"),
 ItemConfirm(text="Night"),
 ItemConfirm(text="Solo"),
 ItemConfirm(text="Phobos"),
 ItemConfirm(text="Diamond"),
 ItemConfirm(text="Sirena"),
 ItemConfirm(text="Red music"),
 ItemConfirm(text="Allergio"),
 ItemConfirm(text="Magic"),
 ItemConfirm(text="Tic-tac"),
],
 buttons=[
 MDFlatButton(
 text="CANCEL", text_color=self.theme_cls.primary_color
),
 MDFlatButton(
 text="OK", text_color=self.theme_cls.primary_color
),
],
)
 self.dialog.open()

Example().run()

```



`items` is an `ListProperty` and defaults to `[]`.

**width\_offset**

Dialog offset from device width.

`width_offset` is an `NumericProperty` and defaults to `dp(48)`.

**type**

Dialog type. Available option are `'alert'`, `'simple'`, `'confirmation'`, `'custom'`.

`type` is an `OptionProperty` and defaults to `'alert'`.

**content\_cls**

Custom content class.

```
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
```

(continues on next page)

(continued from previous page)

```

from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.dialog import MDDialog

KV = '''
<Content>
 orientation: "vertical"
 spacing: "12dp"
 size_hint_y: None
 height: "120dp"

 MDTextField:
 hint_text: "City"

 MDTextField:
 hint_text: "Street"

MDFloatLayout:

 MDFloatButton:
 text: "ALERT DIALOG"
 pos_hint: {'center_x': .5, 'center_y': .5}
 on_release: app.show_confirmation_dialog()
'''

class Content(BoxLayout):
 pass

class Example(MDApp):
 dialog = None

 def build(self):
 return Builder.load_string(KV)

 def show_confirmation_dialog(self):
 if not self.dialog:
 self.dialog = MDDialog(
 title="Address:",
 type="custom",
 content_cls=Content(),
 buttons=[
 MDFlatButton(
 text="CANCEL", text_color=self.theme_cls.primary_color
),
 MDFlatButton(
 text="OK", text_color=self.theme_cls.primary_color
),
],
)

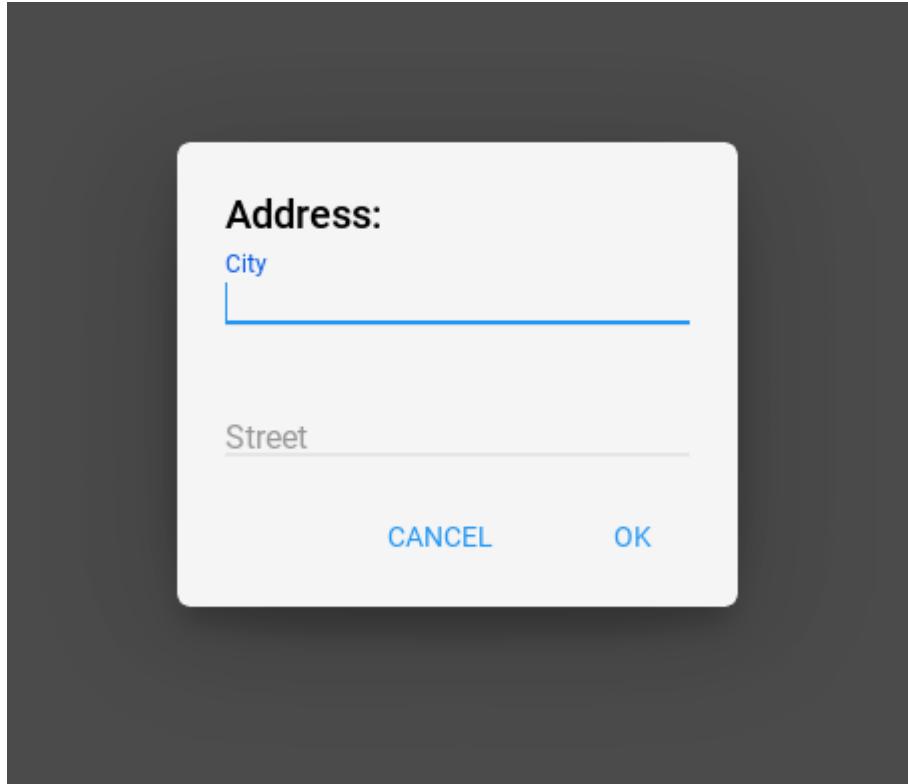
```

(continues on next page)

(continued from previous page)

```
self.dialog.open()
```

```
Example().run()
```



`content_cls` is an `ObjectProperty` and defaults to '`None`'.

#### `md_bg_color`

Background color in the format (r, g, b, a).

`md_bg_color` is an `ColorProperty` and defaults to `None`.

`update_width(self, *args)`

`update_height(self, *args)`

`on_open(self)`

`get_normal_height(self)`

`edit_padding_for_item(self, instance_item)`

`create_items(self)`

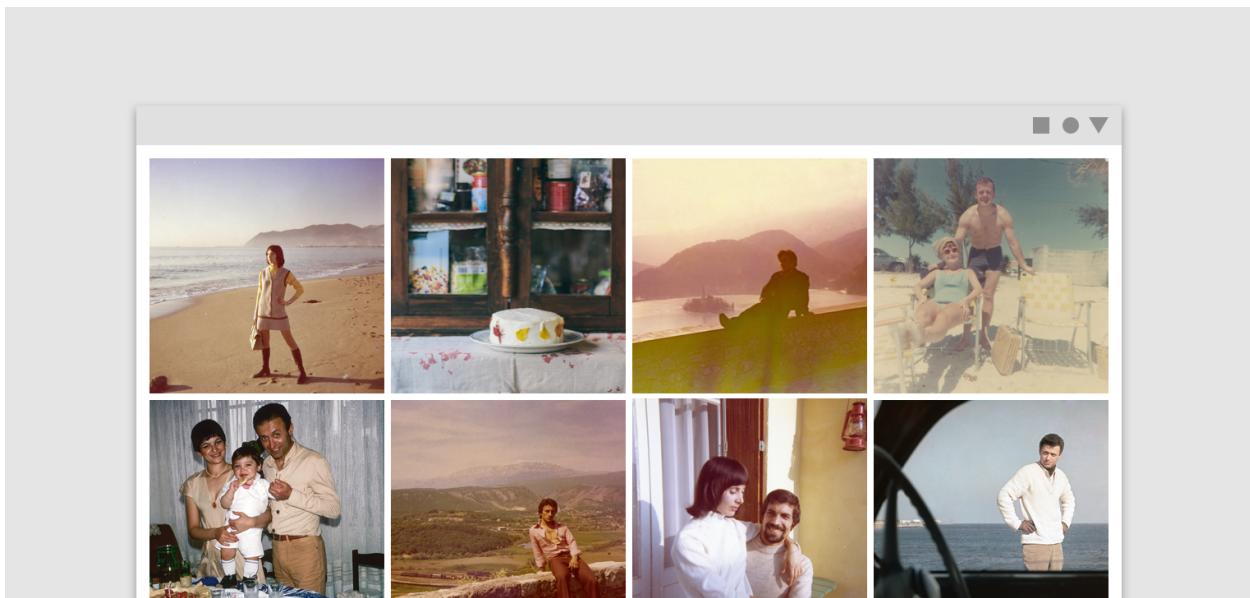
`create_buttons(self)`

## 2.3.24 Image List

**See also:**

Material Design spec, Image lists

**Image lists display a collection of images in an organized grid.**



KivyMD provides the following tile classes for use:

- *SmartTileWithStar*
- *SmartTileWithLabel*

### SmartTileWithStar

```
from kivymd.app import MDApp
from kivy.lang import Builder
```

```
KV = '''
<MyTile@SmartTileWithStar>
 size_hint_y: None
 height: "240dp"
```

ScrollView:

```
MDGridLayout:
 cols: 3
 adaptive_height: True
 padding: dp(4), dp(4)
 spacing: dp(4)
```

(continues on next page)

(continued from previous page)

```
MyTile:
 stars: 5
 source: "cat-1.jpg"

MyTile:
 stars: 5
 source: "cat-2.jpg"

MyTile:
 stars: 5
 source: "cat-3.jpg"
...

class MyApp(MDApp):
 def build(self):
 return Builder.load_string(KV)

MyApp().run()
```

## SmartTileWithLabel

```
from kivymd.app import MDApp
from kivy.lang import Builder

KV = ''''
<MyTile@SmartTileWithLabel>
 size_hint_y: None
 height: "240dp"

ScrollView:

 MDGridLayout:
 cols: 3
 adaptive_height: True
 padding: dp(4), dp(4)
 spacing: dp(4)

 MyTile:
 source: "cat-1.jpg"
 text: "[size=26]Cat 1[/size]\n[size=14]cat-1.jpg[/size]"

 MyTile:
 source: "cat-2.jpg"
 text: "[size=26]Cat 2[/size]\n[size=14]cat-2.jpg[/size]"
 tile_text_color: app.theme_cls.accent_color
```

(continues on next page)

(continued from previous page)

```

MyTile:
 source: "cat-3.jpg"
 text: "[size=26][color=#ffffffff]Cat 3[/color][/size]\n[size=14]cat-3.jpg[/
 size]"
 tile_text_color: app.theme_cls.accent_color
...

```

```

class MyApp(MDApp):
 def build(self):
 return Builder.load_string(KV)

MyApp().run()

```



### API - kivymd.uix.imagelist

```
class kivymd.uix.imagelist.SmartTile(**kwargs)
A tile for more complex needs.
```

Includes an image, a container to place overlays and a box that can act as a header or a footer, as described in the Material Design specs.

#### **box\_color**

Sets the color and opacity for the information box.

*box\_color* is a [ColorProperty](#) and defaults to *(0, 0, 0, 0.5)*.

#### **box\_position**

Determines whether the information box acts as a header or footer to the image. Available are options: *'footer'*, *'header'*.

*box\_position* is a [OptionProperty](#) and defaults to *'footer'*.

#### **lines**

Number of lines in the *header/footer*. As per *Material Design specs*, only 1 and 2 are valid values. Available are options: 1, 2.

*lines* is a [OptionProperty](#) and defaults to 1.

**overlap**

Determines if the *header/footer* overlaps on top of the image or not.

*overlap* is a [BooleanProperty](#) and defaults to *True*.

**source**

Path to tile image. See [source](#).

*source* is a [StringProperty](#) and defaults to ''.

**reload(self)**

**class kivymd.uix.imagelist.SmartTileWithLabel(\*\*kwargs)**

A tile for more complex needs.

Includes an image, a container to place overlays and a box that can act as a header or a footer, as described in the Material Design specs.

**font\_style**

Tile font style.

*font\_style* is a [StringProperty](#) and defaults to ‘Caption’.

**tile\_text\_color**

Tile text color in `rgba` format.

*tile\_text\_color* is a [ColorProperty](#) and defaults to *(1, 1, 1, 1)*.

**text**

Determines the text for the box *footer/header*.

*text* is a [StringProperty](#) and defaults to ''.

**class kivymd.uix.imagelist.SmartTileWithStar(\*\*kwargs)**

A tile for more complex needs.

Includes an image, a container to place overlays and a box that can act as a header or a footer, as described in the Material Design specs.

**stars**

Tile stars.

*stars* is a [NumericProperty](#) and defaults to *1*.

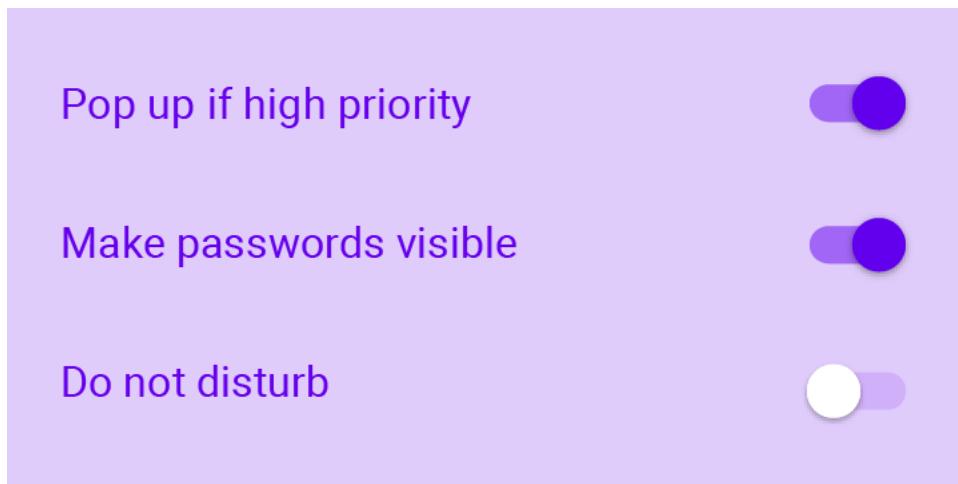
**on\_stars(self, \*args)**

### 2.3.25 Selection Controls

**See also:**

[Material Design spec, Selection controls](#)

**Selection controls allow the user to select options.**



KivyMD provides the following selection controls classes for use:

- *MDCheckbox*
- *MDSwitch*

### MDCheckbox

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDFloatLayout:

 MDCheckbox:
 size_hint: None, None
 size: "48dp", "48dp"
 pos_hint: {'center_x': .5, 'center_y': .5}
'''

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```

---

**Note:** Be sure to specify the size of the checkbox. By default, it is (dp(48), dp(48)), but the ripple effect takes up all the available space.

## Control state

### MDCheckbox:

```
 on_active: app.on_checkbox_active(*args)
```

```
def on_checkbox_active(self, checkbox, value):
 if value:
 print('The checkbox', checkbox, 'is active', 'and', checkbox.state, 'state')
 else:
 print('The checkbox', checkbox, 'is inactive', 'and', checkbox.state, 'state')
```

## MDCheckbox with group

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<Check@MDCheckbox>:
 group: 'group'
 size_hint: None, None
 size: dp(48), dp(48)

MDFloatLayout:

 Check:
 active: True
 pos_hint: {'center_x': .4, 'center_y': .5}

 Check:
 pos_hint: {'center_x': .6, 'center_y': .5}
'''

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```

## MDSwitch

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDFloatLayout:

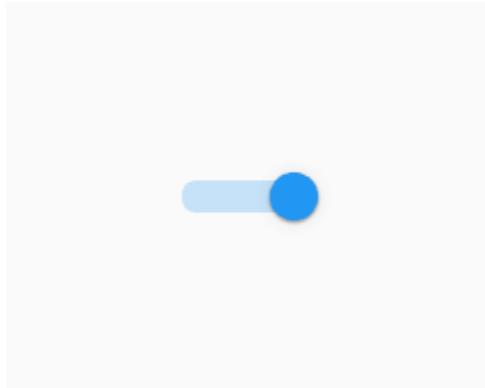
 MDSwitch:
 pos_hint: {'center_x': .5, 'center_y': .5}
'''

class Test(MDApp):
 def build(self):
 return Builder.load_string(KV)

Test().run()
```

**Note:** For `MDSwitch` size is not required. By default it is (`dp(36)`, `dp(48)`), but you can increase the width if you want.

```
MDSwitch:
 width: dp(64)
```



**Note:** Control state of `MDSwitch` same way as in `MDCheckbox`.

**API - kivymd.uix.selectioncontrol****class kivymd.uix.selectioncontrol.MDCheckbox(\*\*kwargs)**

Class implements a circular ripple effect.

**active**

Indicates if the checkbox is active or inactive.

*active* is a `BooleanProperty` and defaults to `False`.**checkbox\_icon\_normal**

Background icon of the checkbox used for the default graphical representation when the checkbox is not pressed.

*checkbox\_icon\_normal* is a `StringProperty` and defaults to ‘checkbox-blank-outline’.**checkbox\_icon\_down**

Background icon of the checkbox used for the default graphical representation when the checkbox is pressed.

*checkbox\_icon\_down* is a `StringProperty` and defaults to ‘checkbox-marked’.**radio\_icon\_normal**

Background icon (when using the group option) of the checkbox used for the default graphical representation when the checkbox is not pressed.

*radio\_icon\_normal* is a `StringProperty` and defaults to ‘checkbox-blank-circle-outline’.**radio\_icon\_down**

Background icon (when using the group option) of the checkbox used for the default graphical representation when the checkbox is pressed.

*radio\_icon\_down* is a `StringProperty` and defaults to ‘checkbox-marked-circle’.**selected\_color**Selected color in `rgba` format.*selected\_color* is a `ColorProperty` and defaults to `None`.**unselected\_color**Unelected color in `rgba` format.*unselected\_color* is a `ColorProperty` and defaults to `None`.**disabled\_color**Disabled color in `rgba` format.*disabled\_color* is a `ColorProperty` and defaults to `None`.**update\_primary\_color(self, instance, value)****update\_icon(self, \*args)****update\_color(self, \*args)****on\_state(self, \*args)****on\_active(self, \*args)****class kivymd.uix.selectioncontrol.MDSwitch(\*\*kwargs)**This `mixin` class provides `Button` behavior. Please see the `button behaviors` module documentation for more information.**Events***on\_press* Fired when the button is pressed.

***on\_release*** Fired when the button is released (i.e. the touch/click that pressed the button goes away).

#### **active**

Indicates if the switch is active or inactive.

`active` is a `BooleanProperty` and defaults to `False`.

#### **thumb\_color**

Get thumb color `rgba` format.

`thumb_color` is an `AliasProperty` and property is readonly.

#### **thumb\_color\_disabled**

Get thumb color disabled `rgba` format.

`thumb_color_disabled` is an `AliasProperty` and property is readonly.

#### **thumb\_color\_down**

Get thumb color down `rgba` format.

`thumb_color_down` is an `AliasProperty` and property is readonly.

#### **theme\_thumb\_color**

Thumb color scheme name

`theme_thumb_color` is an `OptionProperty` and defaults to `Primary`.

#### **theme\_thumb\_down\_color**

Thumb Down color scheme name

`theme_thumb_down_color` is an `OptionProperty` and defaults to `Primary`.

#### **on\_size(self, \*args)**

### 2.3.26 Card

#### See also:

Material Design spec, Cards

**Cards contain content and actions about a single subject.**

KivyMD provides the following card classes for use:

- `MDCard`
- `MDCardSwipe`

---

**Note:** `MDCard` inherited from `BoxLayout`. You can use all parameters and attributes of the `BoxLayout` class in the `MDCard` class.

---

**MDCard**

```
from kivy.lang import Builder

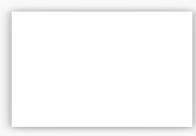
from kivymd.app import MDApp

KV = '''
MDScreen:

 MDCard:
 size_hint: None, None
 size: "280dp", "180dp"
 pos_hint: {"center_x": .5, "center_y": .5}
 ...

class TestCard(MDApp):
 def build(self):
 return Builder.load_string(KV)

TestCard().run()
```

**Add content to card:**

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

 MDCard:
 orientation: "vertical"
 padding: "8dp"
 size_hint: None, None
 size: "280dp", "180dp"
 pos_hint: {"center_x": .5, "center_y": .5}

 MDLabel:
 text: "Title"
 theme_text_color: "Secondary"
 size_hint_y: None
 height: self.texture_size[1]

 MDSeparator:
```

(continues on next page)

(continued from previous page)

```

height: "1dp"

MDLabel:
 text: "Body"
...
.

class TestCard(MDApp):
 def build(self):
 return Builder.load_string(KV)

TestCard().run()

```



## MDCardSwipe

To create a card with *swipe-to-delete* behavior, you must create a new class that inherits from the `MDCardSwipe` class:

```

<SwipeToDeleteItem>:
 size_hint_y: None
 height: content.height

MDCardSwipeLayerBox:

MDCardSwipeFrontBox:

 OneLineListItem:
 id: content
 text: root.text
 _no_ripple_effect: True

```

```

class SwipeToDeleteItem(MDCardSwipe):
 text = StringProperty()

```



**End full code**

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.card import MDCardSwipe

KV = '''
<SwipeToDeleteItem>:
 size_hint_y: None
 height: content.height

 MDCardSwipeLayerBox:
 # Content under the card.

 MDCardSwipeFrontBox:

 # Content of card.
 OneLineListItem:
 id: content
 text: root.text
 _no_ripple_effect: True

MDScreen:

 MDBBoxLayout:
 orientation: "vertical"
 spacing: "10dp"

 MDToolbar:
 elevation: 10
 title: "MDCardSwipe"

 ScrollView:
 scroll_timeout : 100

 MDList:
 id: md_list
 padding: 0
 ...

class SwipeToDeleteItem(MDCardSwipe):
 "Card with `swipe-to-delete` behavior."

 text = StringProperty()

class TestCard(MDApp):
 def __init__(self, **kwargs):
```

(continues on next page)

(continued from previous page)

```

super().__init__(**kwargs)
self.screen = Builder.load_string(KV)

def build(self):
 return self.screen

def on_start(self):
 "Creates a list of cards."

 for i in range(20):
 self.screen.ids.md_list.add_widget(
 SwipeToDeleteItem(text=f"One-line item {i}")
)

```

TestCard().run()

### Binding a swipe to one of the sides of the screen

```

<SwipeToDeleteItem>:
 # By default, the parameter is "left"
 anchor: "right"

```

---

**Note:** You cannot use the left and right swipe at the same time.

---

### Swipe behavior

```

<SwipeToDeleteItem>:
 # By default, the parameter is "hand"
 type_swipe: "hand"

```

```

<SwipeToDeleteItem>:
 type_swipe: "auto"

```

### Removing an item using the type\_swipe = "auto" parameter

The map provides the `MDCardSwipe.on_swipe_complete` event. You can use this event to remove items from a list:

```
<SwipeToDeleteItem>:
 on_swipe_complete: app.on_swipe_complete(root)
```

```
def on_swipe_complete(self, instance):
 self.screen.ids.md_list.remove_widget(instance)
```

End full code

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.card import MDCardSwipe

KV = '''
<SwipeToDeleteItem>:
 size_hint_y: None
 height: content.height
 type_swipe: "auto"
 on_swipe_complete: app.on_swipe_complete(root)

 MDCardSwipeLayerBox:

 MDCardSwipeFrontBox:

 OneLineListItem:
 id: content
 text: root.text
 _no_ripple_effect: True

MDScreen:

 MDBBoxLayout:
 orientation: "vertical"
 spacing: "10dp"

 MDToolbar:
 elevation: 10
 title: "MDCardSwipe"

 ScrollView:

 MDList:
 id: md_list
 padding: 0
'''
```

(continues on next page)

(continued from previous page)

```

class SwipeToDeleteItem(MDCardSwipe):
 text = StringProperty()

class TestCard(MDApp):
 def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)

 def build(self):
 return self.screen

 def on_swipe_complete(self, instance):
 self.screen.ids.md_list.remove_widget(instance)

 def on_start(self):
 for i in range(20):
 self.screen.ids.md_list.add_widget(
 SwipeToDeleteItem(text=f"One-line item {i}")
)

TestCard().run()

```

### Add content to the bottom layer of the card

To add content to the bottom layer of the card, use the `MDCardSwipeLayerBox` class.

```

<SwipeToDeleteItem>:

 MDCardSwipeLayerBox:
 padding: "8dp"

 MDIconButton:
 icon: "trash-can"
 pos_hint: {"center_y": .5}
 on_release: app.remove_item(root)

```

**End full code**

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.card import MDCardSwipe

KV = '''
<SwipeToDeleteItem>:
 size_hint_y: None
 height: content.height

 MDCardSwipeLayerBox:
 padding: "8dp"

 MDIconButton:
 icon: "trash-can"
 pos_hint: {"center_y": .5}
 on_release: app.remove_item(root)

 MDCardSwipeFrontBox:

 OneLineListItem:
 id: content
 text: root.text
 _no_ripple_effect: True

MDScreen:

 MDBBoxLayout:
 orientation: "vertical"
 spacing: "10dp"

 MDToolbar:
 elevation: 10
 title: "MDCardSwipe"

 ScrollView:

 MDList:
 id: md_list
 padding: 0
 ...

class SwipeToDeleteItem(MDCardSwipe):
 text = StringProperty()

class TestCard(MDApp):
```

(continues on next page)

(continued from previous page)

```

def __init__(self, **kwargs):
 super().__init__(**kwargs)
 self.screen = Builder.load_string(KV)

def build(self):
 return self.screen

def remove_item(self, instance):
 self.screen.ids.md_list.remove_widget(instance)

def on_start(self):
 for i in range(20):
 self.screen.ids.md_list.add_widget(
 SwipeToDeleteItem(text=f"One-line item {i}")
)

```

TestCard().run()

## Focus behavior

```

MDCard:
 focus_behavior: True

```

## Ripple behavior

```

MDCard:
 ripple_behavior: True

```

## End full code

```

from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<StarButton@MDIconButton>
 icon: "star"
 on_release: self.icon = "star-outline" if self.icon == "star" else "star"

```

MDScreen:

(continues on next page)

(continued from previous page)

```
MDCard:
 orientation: "vertical"
 size_hint: .5, None
 height: box_top.height + box_bottom.height
 focus_behavior: True
 ripple_behavior: True
 pos_hint: {"center_x": .5, "center_y": .5}

MDBBoxLayout:
 id: box_top
 spacing: "20dp"
 adaptive_height: True

FitImage:
 source: "/Users/macbookair/album.jpeg"
 size_hint: .3, None
 height: text_box.height

MDBBoxLayout:
 id: text_box
 orientation: "vertical"
 adaptive_height: True
 spacing: "10dp"
 padding: 0, "10dp", "10dp", "10dp"

MDLabel:
 text: "Ride the Lightning"
 theme_text_color: "Primary"
 font_style: "H5"
 bold: True
 size_hint_y: None
 height: self.texture_size[1]

MDLabel:
 text: "July 27, 1984"
 size_hint_y: None
 height: self.texture_size[1]
 theme_text_color: "Primary"

MDSeparator:

MDBBoxLayout:
 id: box_bottom
 adaptive_height: True
 padding: "10dp", 0, 0, 0

MDLabel:
 text: "Rate this album"
 size_hint_y: None
 height: self.texture_size[1]
 pos_hint: {"center_y": .5}
```

(continues on next page)

(continued from previous page)

```

 theme_text_color: "Primary"

 StarButton:
 StarButton:
 StarButton:
 StarButton:
 StarButton:
 ...
 ...

class Test(MDApp):
 def build(self):
 self.theme_cls.theme_style = "Dark"
 return Builder.load_string(KV)

Test().run()

```

**API - kivymd.uix.card****class kivymd.uix.card.MDSeparator(\*\*kwargs)**

A separator line.

**color**

Separator color in rgba format.

*color* is a [ColorProperty](#) and defaults to *None*.**on\_orientation(self, \*args)****class kivymd.uix.card.MDCard(\*\*kwargs)***FakeRectangularElevationBehavior* is a shadow mockup for widgets. Improves performance using cached images inside `kivymd/images` dir

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```

class NewWidget(
 ThemableBehavior,
 FakeCircularElevationBehavior,
 SpecificBackgroundColorBehavior,
 # here you add the other front end classes for the widget front_end,
):
 [...]

```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

*FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
 MDCard,
 FakeRectangularElevationBehavior
):
 [...]
```

---

**Note:** About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

---

#### **focus\_behavior**

Using focus when hovering over a card.

*focus\_behavior* is a `BooleanProperty` and defaults to `False`.

#### **ripple\_behavior**

Use ripple effect for card.

*ripple\_behavior* is a `BooleanProperty` and defaults to `False`.

#### **elevation**

Elevation value.

*elevation* is an `NumericProperty` and defaults to 1.

`update_md_bg_color(self, instance, value)`

`on_ripple_behavior(self, instance, value)`

```
class kivymd.uix.card.MDCardSwipe(**kw)
```

### Events

`on_swipe_complete` Called when a swipe of card is completed.

#### **open\_progress**

Percent of visible part of side panel. The percent is specified as a floating point number in the range 0-1. 0.0 if panel is closed and 1.0 if panel is opened.

*open\_progress* is a `NumericProperty` and defaults to `0.0`.

#### **opening\_transition**

The name of the animation transition type to use when animating to the `state 'opened'`.

*opening\_transition* is a `StringProperty` and defaults to `'out_cubic'`.

#### **closing\_transition**

The name of the animation transition type to use when animating to the `state 'closed'`.

*closing\_transition* is a `StringProperty` and defaults to `'out_sine'`.

#### **anchor**

Anchoring screen edge for card. Available options are: `'left'`, `'right'`.

*anchor* is a `OptionProperty` and defaults to `left`.

#### **swipe\_distance**

The distance of the swipe with which the movement of navigation drawer begins.

*swipe\_distance* is a `NumericProperty` and defaults to `50`.

**opening\_time**

The time taken for the card to slide to the `state` ‘open’.

`opening_time` is a `NumericProperty` and defaults to `0.2`.

**state**

Detailed state. Sets before `state`. Bind to `state` instead of `status`. Available options are: ‘closed’, ‘opened’.

`status` is a `OptionProperty` and defaults to ‘closed’.

**max\_swipe\_x**

If, after the events of `on_touch_up` card position exceeds this value - will automatically execute the method `open_card`, and if not - will automatically be `close_card` method.

`max_swipe_x` is a `NumericProperty` and defaults to `0.3`.

**max\_opened\_x**

The value of the position the card shifts to when `type_swipe` s set to ‘hand’.

`max_opened_x` is a `NumericProperty` and defaults to `100dp`.

**type\_swipe**

Type of card opening when swipe. Shift the card to the edge or to a set position `max_opened_x`. Available options are: ‘auto’, ‘hand’.

`type_swipe` is a `OptionProperty` and defaults to `auto`.

**add\_widget(self, widget, index=0, canvas=None)**

Add a new widget as a child of this widget.

**Parameters**

**widget: Widget** Widget to add to our list of children.

**index: int, defaults to 0** Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

**canvas: str, defaults to None** Canvas to add widget’s canvas to. Can be ‘before’, ‘after’ or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

**on\_swipe\_complete(self, \*args)**

Called when a swipe of card is completed.

**on\_anchor(self, instance, value)****on\_open\_progress(self, instance, value)****on\_touch\_move(self, touch)**

Receive a touch move event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

**on\_touch\_up(self, touch)**

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

**on\_touch\_down(self, touch)**

Receive a touch down event.

**Parameters**

`touch: MotionEvent class` Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

**Returns** bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

**complete\_swipe(self)**

**open\_card(self)**

**close\_card(self)**

**class kivymd.uix.card.MDCardSwipeFrontBox(\*\*kwargs)**

*FakeRectangularElevationBehavior` is a shadow mockup for widgets. Improves performance using cached images inside `kivymd.images dir*

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
 ThemableBehavior,
 FakeCircularElevationBehavior,
 SpecificBackgroundColorBehavior,
 # here you add the other front end classes for the widget front_end,
):
 [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

*FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
 MDCard,
 FakeRectangularElevationBehavior
):
 [...]
```

---

**Note:** About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

---

```
class kivymd.uix.card.MDCardSwipeLayerBox(**kwargs)
 Box layout class. See module documentation for more information.
```

### 2.3.27 Bottom Sheet

See also:

Material Design spec, Sheets: bottom

**Bottom sheets are surfaces containing supplementary content that are anchored to the bottom of the screen.**

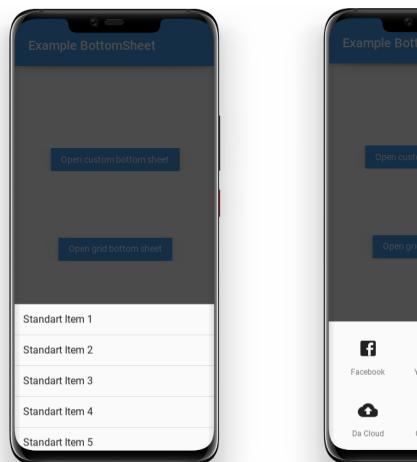


Share

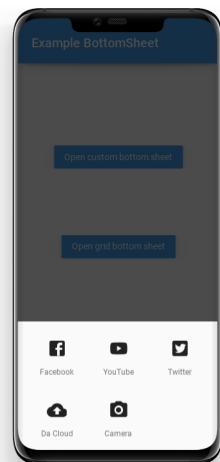


Get link

Two classes are available to you [MDListBottomSheet](#) and [MDGridBottomSheet](#) for standard bottom sheets dialogs:



**MDListBottomSheet**



**MDGridBottomSheet**

### Usage MDListBottomSheet

```
from kivy.lang import Builder

from kivymd.toast import toast
from kivymd.uix.bottomsheet import MDListBottomSheet
from kivymd.app import MDApp

KV = '''
Screen:

 MDToolbar:
 title: "Example BottomSheet"
 pos_hint: {"top": 1}
 elevation: 10

 MDRaisedButton:
 text: "Open list bottom sheet"
 on_release: app.show_example_list_bottom_sheet()
 pos_hint: {"center_x": .5, "center_y": .5}
'''

class Example(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def callback_for_menu_items(self, *args):
 toast(args[0])

 def show_example_list_bottom_sheet(self):
 bottom_sheet_menu = MDListBottomSheet()
 for i in range(1, 11):
 bottom_sheet_menu.add_item(
 f"Standart Item {i}",
 lambda x, y=i: self.callback_for_menu_items(
 f"Standart Item {y}"
),
)
 bottom_sheet_menu.open()

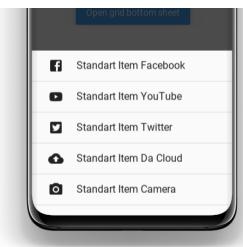
Example().run()
```

The `add_item` method of the `MDListBottomSheet` class takes the following arguments:

`text` - element text;

`callback` - function that will be called when clicking on an item;

There is also an optional argument `icon`, which will be used as an icon to the left of the item:



Using the `MDGridBottomSheet` class is similar to using the `MDListBottomSheet` class:

```
from kivy.lang import Builder

from kivymd.toast import toast
from kivymd.uix.bottomsheet import MDGridBottomSheet
from kivymd.app import MDApp

KV = '''
Screen:

 MDToolbar:
 title: 'Example BottomSheet'
 pos_hint: {"top": 1}
 elevation: 10

 MDRaisedButton:
 text: "Open grid bottom sheet"
 on_release: app.show_example_grid_bottom_sheet()
 pos_hint: {"center_x": .5, "center_y": .5}
'''

class Example(MDApp):
 def build(self):
 return Builder.load_string(KV)

 def callback_for_menu_items(self, *args):
 toast(args[0])

 def show_example_grid_bottom_sheet(self):
 bottom_sheet_menu = MDGridBottomSheet()
 data = {
 "Facebook": "facebook-box",
 "YouTube": "youtube",
 "Twitter": "twitter-box",
 "Da Cloud": "cloud-upload",
 "Camera": "camera",
 }
 for item in data.items():
 bottom_sheet_menu.add_item(
 item[0],
 lambda x, y=item[0]: self.callback_for_menu_items(y),
)

```

(continues on next page)

(continued from previous page)

```
 icon_src=item[1],
)
bottom_sheet_menu.open()
```

```
Example().run()
```



You can use custom content for bottom sheet dialogs:

```
from kivy.lang import Builder
from kivy.factory import Factory

from kivymd.uix.bottomsheet import MDCustomBottomSheet
from kivymd.app import MDApp

KV = ''''
<ItemForCustomBottomSheet@OneLineIconListItem>
 on_press: app.custom_sheet.dismiss()
 icon: """

 IconLeftWidget:
 icon: root.icon

<ContentCustomSheet@BoxLayout>:
 orientation: "vertical"
 size_hint_y: None
 height: "400dp"

 MDToolbar:
 title: 'Custom bottom sheet:'

 ScrollView:

 MDGridLayout:
 cols: 1
 adaptive_height: True

 ItemForCustomBottomSheet:
 icon: "page-previous"
 text: "Preview"
```

(continues on next page)

(continued from previous page)

```

ItemForCustomBottomSheet:
 icon: "exit-to-app"
 text: "Exit"

Screen:

MDToolbar:
 title: 'Example BottomSheet'
 pos_hint: {"top": 1}
 elevation: 10

MDRaisedButton:
 text: "Open custom bottom sheet"
 on_release: app.show_example_custom_bottom_sheet()
 pos_hint: {"center_x": .5, "center_y": .5}
...
```

```

```

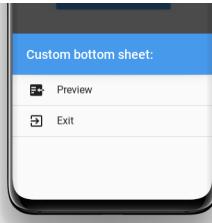
class Example(MDApp):
    custom_sheet = None

    def build(self):
        return Builder.load_string(KV)

    def show_example_custom_bottom_sheet(self):
        self.custom_sheet = MDCustomBottomSheet(screen=Factory.ContentCustomSheet())
        self.custom_sheet.open()

Example().run()

```



Note: When you use the `MDCustomBottomSheet` class, you must specify the height of the user-defined content exactly, otherwise `dp(100)` heights will be used for your `ContentCustomSheet` class:

```

<ContentCustomSheet@BoxLayout>:
    orientation: "vertical"
    size_hint_y: None
    height: "400dp"

```

Note: The height of the bottom sheet dialog will never exceed half the height of the screen!

API - kivymd.uix.bottomsheet

```
class kivymd.uix.bottomsheet.MDBottomSheet(**kwargs)
```

ModalView class. See module documentation for more information.

Events

on_pre_open: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

on_open: Fired when the ModalView is opened.

on_pre_dismiss: Fired before the ModalView is closed.

on_dismiss: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events *on_pre_open* and *on_pre_dismiss*.

Changed in version 2.0.0: Added property ‘overlay_color’.

background

Private attribute.

duration_opening

The duration of the bottom sheet dialog opening animation.

duration_opening is an `NumericProperty` and defaults to *0.15*.

duration_closing

The duration of the bottom sheet dialog closing animation.

duration_closing is an `NumericProperty` and defaults to *0.15*.

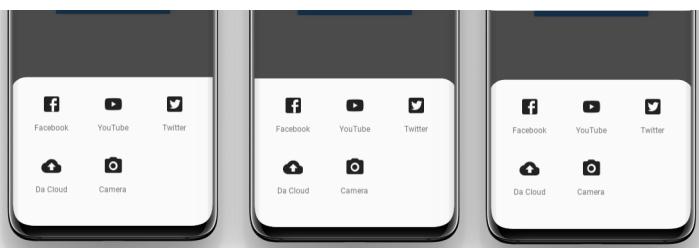
radius

The value of the rounding of the corners of the dialog.

radius is an `NumericProperty` and defaults to *25*.

radius_from

Sets which corners to cut from the dialog. Available options are: (“*top_left*”, “*top_right*”, “*top*”, “*bottom_right*”, “*bottom_left*”, “*bottom*”).



radius_from is an `OptionProperty` and defaults to *None*.

animation

Whether to use animation for opening and closing of the bottomsheet or not.

animation is an `BooleanProperty` and defaults to *False*.

bg_color

Dialog background color in rgba format.

bg_color is an [ColorProperty](#) and defaults to `[]`.

value_transparent

Background transparency value when opening a dialog.

value_transparent is an [ColorProperty](#) and defaults to `[0, 0, 0, 0.8]`.

open(self, *args)

Show the view window from the `attach_to` widget. If set, it will attach to the nearest window. If the widget is not attached to any window, the view will attach to the global `Window`.

When the view is opened, it will be faded in with an animation. If you don't want the animation, use:

```
view.open(animation=False)
```

add_widget(self, widget, index=0, canvas=None)

Add a new widget as a child of this widget.

Parameters

widget: Widget Widget to add to our list of children.

index: int, defaults to 0 Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

canvas: str, defaults to None Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

dismiss(self, *args, **kwargs)

Close the view if it is open. If you really want to close the view, whatever the `on_dismiss` event returns, you can use the `force` argument:

```
view = ModalView()
view.dismiss(force=True)
```

When the view is dismissed, it will be faded out before being removed from the parent. If you don't want animation, use:

```
view.dismiss(animation=False)
```

resize_content_layout(self, content, layout, interval=0)**class kivymd.uix.bottomsheet.MDCustomBottomSheet(**kwargs)**

ModalView class. See module documentation for more information.

Events

on_pre_open: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

on_open: Fired when the ModalView is opened.

on_pre_dismiss: Fired before the ModalView is closed.

on_dismiss: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events *on_pre_open* and *on_pre_dismiss*.

Changed in version 2.0.0: Added property ‘overlay_color’.

screen

Custom content.

`screen` is an `ObjectProperty` and defaults to `None`.

`class kivymd.uix.bottomsheet.MDListBottomSheet(**kwargs)`

ModalView class. See module documentation for more information.

Events

on_pre_open: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

on_open: Fired when the ModalView is opened.

on_pre_dismiss: Fired before the ModalView is closed.

on_dismiss: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events *on_pre_open* and *on_pre_dismiss*.

Changed in version 2.0.0: Added property ‘overlay_color’.

sheet_list

`sheet_list` is an `ObjectProperty` and defaults to `None`.

`add_item(self, text, callback, icon=None)`

Parameters

- **`text`** – element text;
- **`callback`** – function that will be called when clicking on an item;
- **`icon`** – which will be used as an icon to the left of the item;

`class kivymd.uix.bottomsheet.GridBottomSheetItem(**kwargs)`

This `mixin` class provides `Button` behavior. Please see the `button behaviors` module documentation for more information.

Events

on_press Fired when the button is pressed.

on_release Fired when the button is released (i.e. the touch/click that pressed the button goes away).

source

Icon path if you use a local image or icon name if you use icon names from a file `kivymd/icon_definitions.py`.

`source` is an `StringProperty` and defaults to ''.

caption

Item text.

`caption` is an `StringProperty` and defaults to ''.

icon_size

Icon size.

`caption` is an `StringProperty` and defaults to '24sp'.

class kivymd.uix.bottomsheet.MDGridBottomSheet(kwargs)**

ModalView class. See module documentation for more information.

Events

`on_pre_open`: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

`on_open`: Fired when the ModalView is opened.

`on_pre_dismiss`: Fired before the ModalView is closed.

`on_dismiss`: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events `on_pre_open` and `on_pre_dismiss`.

Changed in version 2.0.0: Added property 'overlay_color'.

`add_item(self, text, callback, icon_src)`

Parameters

- **`text`** – element text;
- **`callback`** – function that will be called when clicking on an item;
- **`icon_src`** – icon item;

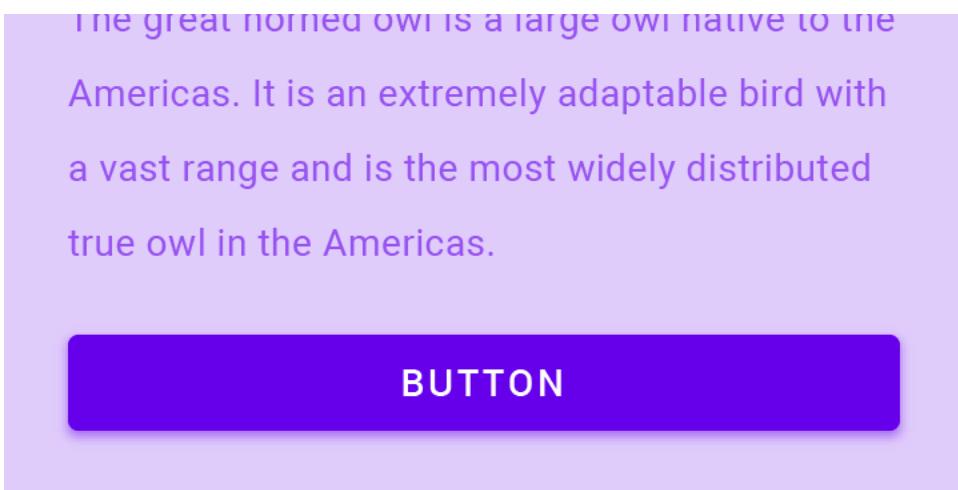
2.3.28 Button

See also:

Material Design spec, Buttons

Material Design spec, Buttons: floating action button

Buttons allow users to take actions, and make choices, with a single tap.



KivyMD provides the following button classes for use:

- [MDIconButton](#)
- [MDFloatingActionButton](#)
- [MDFlatButton](#)
- [MDRaisedButton](#)
- [MDRectangleFlatButton](#)
- [MDRectangleFlatIconButton](#)
- [MDRoundFlatButton](#)
- [MDRoundFlatIconButton](#)
- [MDFillRoundFlatButton](#)
- [MDTextButton](#)
- [MDFloatingActionButtonSpeedDial](#)

MDIconButton

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDIconButton:
        icon: "language-python"
        pos_hint: {"center_x": .5, "center_y": .5}
```

(continues on next page)

(continued from previous page)

```
'''  
  
class Example(MDApp):  
    def build(self):  
        return Builder.load_string(KV)  
  
Example().run()
```

The `icon` parameter must have the name of the icon from `kivymd/icon_definitions.py` file.

You can also use custom icons:

```
MDIconButton:  
    icon: "data/logo/kivy-icon-256.png"
```

By default, `MDIconButton` button has a size (dp(48), dp (48)). Use `user_font_size` attribute to resize the button:

```
MDIconButton:  
    icon: "android"  
    user_font_size: "64sp"
```

By default, the color of `MDIconButton` (depending on the style of the application) is black or white. You can change the color of `MDIconButton` as the text color of `MDLabel`:

```
MDIconButton:  
    icon: "android"  
    theme_text_color: "Custom"  
    text_color: app.theme_cls.primary_color
```



MDFloatingActionButton



The above parameters for `MDIconButton` apply to `MDFloatingActionButton`.

To change `MDFloatingActionButton` background, use the `md_bg_color` parameter:

```
MDFloatingActionButton:
    icon: "android"
    md_bg_color: app.theme_cls.primary_color
```



The length of the shadow is controlled by the `elevation_normal` parameter:

```
MDFloatingActionButton:
    icon: "android"
    elevation_normal: 12
```



MDFlatButton

To change the text color of: class:`~MDFlatButton` use the `text_color` parameter:

```
MDFlatButton:
    text: "MDFLATBUTTON"
    theme_text_color: "Custom"
    text_color: 0, 0, 1, 1
```

MDFLATBUTTON MDFLATBUTTON

Or use markup:

```
MDFFlatButton:
    text: "[color=#00ffcc]MDFLATBUTTON[/color]"
```

To specify the font size and font name, use the parameters as in the usual *Kivy* buttons:

```
MDFFlatButton:
    text: "MDFLATBUTTON"
    font_size: "18sp"
    font_name: "path/to/font"
```

MDRaisedButton

This button is similar to the *MDFFlatButton* button except that you can set the background color for *MDRaisedButton*:

```
MDRaisedButton:
    text: "MDRAISEDBUTTON"
    md_bg_color: 1, 0, 1, 1
```

MDRectangleFlatButton

```
MDRectangleFlatButton:
    text: "MDRECTANGLEFLATBUTTON"
    theme_text_color: "Custom"
    text_color: 1, 0, 0, 1
    line_color: 0, 0, 1, 1
```



MDRectangleFlatButton

MDRectangleFlatIconButton



MDRECTANGLEFLATICONBUTTON

Button parameters *MDRectangleFlatButton* are the same as button *MDRectangleFlatButton*:

```
MDRectangleFlatIconButton:
    icon: "android"
    text: "MDRECTANGLEFLATICONBUTTON"
```

(continues on next page)

(continued from previous page)

```
theme_text_color: "Custom"
text_color: 0, 0, 1, 1
line_color: 1, 0, 1, 1
icon_color: 1, 0, 0, 1
```



Without border

```
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
from kivymd.uix.button import MDRectangleFlatButton

class Example(MDApp):
    def build(self):
        screen = MDScreen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="MDRectangleFlatButton",
                icon="language-python",
                line_color=(0, 0, 0, 0),
                pos_hint={"center_x": .5, "center_y": .5},
            )
        )
    return screen

Example().run()
```

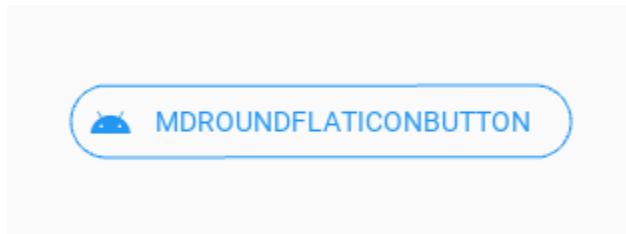
MDRectangleFlatButton:

```
text: "MDRectangleFlatButton"
icon: "language-python"
line_color: 0, 0, 0, 0
pos_hint: {"center_x": .5, "center_y": .5}
```

MDRoundFlatButton

```
MDRoundFlatButton:
    text: "MDROUNDFLATBUTTON"
    text_color: 0, 1, 0, 1
```

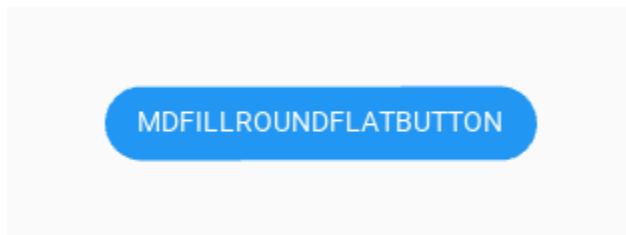
MDRoundFlatButtonIconButton



Button parameters `MDRoundFlatButtonIconButton` are the same as button `MDRoundFlatButton`:

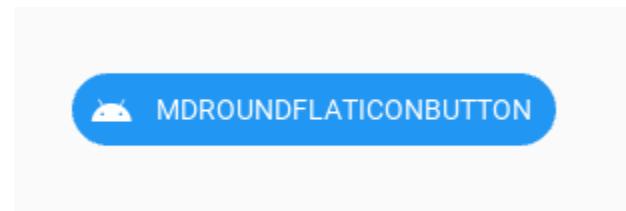
```
MDRoundFlatButtonIconButton:  
    icon: "android"  
    text: "MDROUNDFLATICONBUTTON"
```

MDFillRoundFlatButton



Button parameters `MDFillRoundFlatButton` are the same as button `MDRaisedButton`.

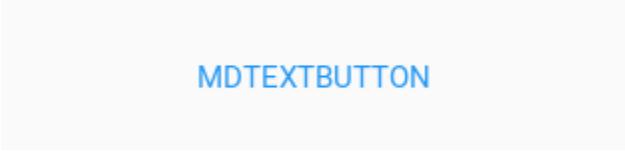
MDFillRoundFlatButtonIconButton



Button parameters `MDFillRoundFlatButtonIconButton` are the same as button `MDRaisedButton`.

Note: Notice that the width of the `MDFillRoundFlatButtonIconButton` button matches the size of the button text.

MDTextButton



MDTEXTBUTTON

```
MDTextButton:  
    text: "MDTEXTBUTTON"  
    custom_color: 0, 1, 0, 1
```

MDFloatingActionButtonSpeedDial

Note: See the full list of arguments in the class *MDFloatingActionButtonSpeedDial*.

```
from kivy.lang import Builder  
  
from kivymd.app import MDApp  
  
KV = ''  
MDScreen:  
  
    MDFloatingActionButtonSpeedDial:  
        data: app.data  
        root_button_anim: True  
...  
  
class Example(MDApp):  
    data = {  
        'Python': 'language-python',  
        'PHP': 'language-php',  
        'C++': 'language-cpp',  
    }  
  
    def build(self):  
        return Builder.load_string(KV)  
  
Example().run()
```

Or without KV Language:

```
from kivymd.uix.screen import MDScreen  
from kivymd.app import MDApp  
from kivymd.uix.button import MDFloatingActionButtonSpeedDial
```

(continues on next page)

(continued from previous page)

```
class Example(MDApp):
    data = {
        'Python': 'language-python',
        'PHP': 'language-php',
        'C++': 'language-cpp',
    }

    def build(self):
        screen = MDScreen()
        speed_dial = MDFloatingActionButtonSpeedDial()
        speed_dial.data = self.data
        speed_dial.root_button_anim = True
        screen.add_widget(speed_dial)
        return screen
```

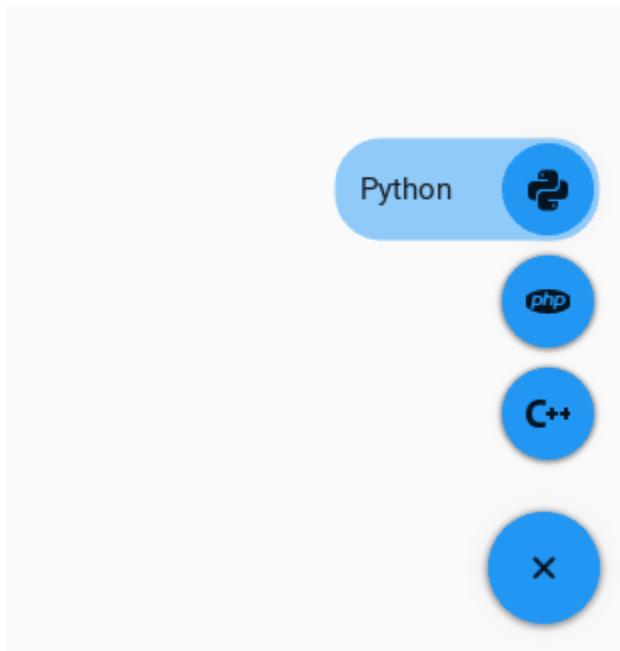
```
Example().run()
```

You can use various types of animation of labels for buttons on the stack:

```
MDFloatingActionButtonSpeedDial:
    hint_animation: True
```

You can set your color values for background, text of buttons etc:

```
MDFloatingActionButtonSpeedDial:
    bg_hint_color: app.theme_cls.primary_light
```



See also:

[See full example](#)

API - kivymd.uix.button

```
class kivymd.uix.button.MDRaisedButton(**kwargs)
    Base class for all rectangular buttons.

    theme_text_color
    update_text_color(self, *args)

class kivymd.uix.button.MDFlatButton(**kwargs)
    Base class for all rectangular buttons.

    md_bg_color

class kivymd.uix.button.MDRectangleFlatButton(**kwargs)
    Base class for all rectangular buttons.

class kivymd.uix.button.MDRectangleFlatButtonIconButton(**kwargs)
    Base class for all rectangular buttons.

    icon
        Button icon.

        icon is an StringProperty and defaults to 'android'.

    icon_color
        Button icon color.

        icon_color is an ColorProperty and defaults to None.

    update_md_bg_color(self, instance, value)
        Called when the application color palette changes.

    set_icon_color(self, interval)
        Sets the icon color if no custom value is specified.

    remove_label(self, interval)

class kivymd.uix.button.MDRoundFlatButton(**kwargs)
    Base class for all rectangular buttons.

    line_width
        Line width for button border.

        line_width is an NumericProperty and defaults to 1.

    line_color
        Line color for button border.

        line_color is an ColorProperty and defaults to None.

    lay_canvas_instructions(self)

class kivymd.uix.button.MDRoundFlatButtonIconButton(**kwargs)
    Base class for all rectangular buttons.

    icon
        Button icon.

        icon is an StringProperty and defaults to 'android'.
```

icon_color
Button icon color.
`icon_color` is an `ColorProperty` and defaults to `None`.

set_icon_color(self, interval)
Sets the icon color if no custom value is specified.

update_md_bg_color(self, instance, value)
Called when the application color palette changes.

on_icon_color(self, instance, value)

remove_label(self, interval)

class kivymd.uix.button.MDFillRoundFlatButton(kwargs)**
Base class for all rectangular buttons.

opposite_colors

set_text_color(self, interval)
Sets the text color if no custom value is specified.

update_md_bg_color(self, instance, value)
Called when the application color palette changes.

on_md_bg_color(self, instance, value)
We override this method, thus prohibiting setting the background color for the button.
Allows to set the background color only in the range from [0.0, 0.0, 0.0, 0.0] to [0.0, 0.0, 0.0, 0.1]. This color is set in the `BasePressedButton` class when the button is pressed and Ignore other custom colors.

class kivymd.uix.button.MDFillRoundFlatIconButton(kwargs)**
Base class for all rectangular buttons.

set_md_bg_color(self, interval)
Checks if a value is set for the `md_bg_color` parameter.

on_md_bg_color(self, instance, value)
We override this method, thus prohibiting setting the background color for the button.
Allows to set the background color only in the range from [0.0, 0.0, 0.0, 0.0] to [0.0, 0.0, 0.0, 0.1]. This color is set in the `BasePressedButton` class when the button is pressed and Ignore other custom colors.

update_md_bg_color(self, instance, value)
Called when the application color palette changes.

update_text_color(self, *args)

set_text_color(self, interval)
Sets the text color if no custom value is specified.

update_icon_color(self, interval)

on_disabled(self, instance, value)
This function hides the shadow when the widget is disabled. It sets the shadow to `0`.

set_icon_color(self, interval)
Sets the icon color if no custom value is specified.

class kivymd.uix.button.MDIconButton(kwargs)**
Base class for all round buttons, bringing in the appropriate on-touch behavior

icon
Button icon.

icon is an `StringProperty` and defaults to ‘checkbox-blank-circle’.

set_size(self, interval)

Sets the custom icon size if the value of the `user_font_size` attribute is not zero. Otherwise, the icon size is set to (48, 48).

update_md_bg_color(self, instance, value)

Called when the application color palette changes.

class kivymd.uix.button.MDFloatingActionButton(kwargs)**

Base class for all round buttons, bringing in the appropriate on-touch behavior

icon

Button icon.

icon is an `StringProperty` and defaults to ‘android’.

update_text_color(self, *args)

set_md_bg_color(self, interval)

Checks if a value is set for the `md_bg_color` parameter.

set_size(self, interval)

on_touch_down(self, touch)

Receive a touch down event.

Parameters

touch: MotionEvent class Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

Returns bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

on_touch_move(self, touch)

Receive a touch move event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

on_touch_up(self, touch)

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

class kivymd.uix.button.MDTextButton(kwargs)**

This `mixin` class provides `Button` behavior. Please see the `button behaviors` module documentation for more information.

Events

on_press Fired when the button is pressed.

on_release Fired when the button is released (i.e. the touch/click that pressed the button goes away).

color

Button color in (r, g, b, a) format.

color is an `ColorProperty` and defaults to `None`.

color_disabled

Button color disabled in (r, g, b, a) format.

color_disabled is an `ColorProperty` and defaults to `None`.

```
animation_label(self)
```

```
on_press(self, *args)
```

```
on_md_bg_color(self, instance, value)
```

```
on_disabled(self, instance, value)
```

This function hides the shadow when the widget is disabled. It sets the shadow to *0*.

```
class kivymd.uix.button.MDFloatingActionButtonSpeedDial(**kwargs)
```

Events

on_open Called when a stack is opened.

on_close Called when a stack is closed.

icon

Root button icon name.

icon is a **StringProperty** and defaults to ‘plus’.

anchor

Stack anchor. Available options are: ‘right’.

anchor is a **OptionProperty** and defaults to ‘right’.

callback

Custom callback.

```
MDFloatingActionButtonSpeedDial:
    callback: app.callback
```

```
def callback(self, instance):
    print(instance.icon)
```

callback is a **ObjectProperty** and defaults to *None*.

label_text_color

Floating text color in (r, g, b, a) format.

label_text_color is a **ColorProperty** and defaults to [0, 0, 0, 1].

data

Must be a dictionary

```
{  
    'name-icon': 'Text label',  
    ...,  
    ...,  
}
```

right_pad

If *True*, the button will increase on the right side by 2.5 pixels if the *hint_animation* parameter equal to *True*.

False

True

right_pad is a `BooleanProperty` and defaults to *False*.

root_button_anim

If True then the root button will rotate 45 degrees when the stack is opened.

root_button_anim is a `BooleanProperty` and defaults to *False*.

opening_transition

The name of the stack opening animation type.

opening_transition is a `StringProperty` and defaults to ‘out_cubic’.

closing_transition

The name of the stack closing animation type.

closing_transition is a `StringProperty` and defaults to ‘out_cubic’.

opening_transition_button_rotation

The name of the animation type to rotate the root button when opening the stack.

opening_transition_button_rotation is a `StringProperty` and defaults to ‘out_cubic’.

closing_transition_button_rotation

The name of the animation type to rotate the root button when closing the stack.

closing_transition_button_rotation is a `StringProperty` and defaults to ‘out_cubic’.

opening_time

Time required for the stack to go to: attr:`state` ‘open’.

opening_time is a `NumericProperty` and defaults to 0.2.

closing_time

Time required for the stack to go to: attr:`state` ‘close’.

closing_time is a `NumericProperty` and defaults to 0.2.

opening_time_button_rotation

Time required to rotate the root button 45 degrees during the stack opening animation.

opening_time_button_rotation is a `NumericProperty` and defaults to 0.2.

closing_time_button_rotation

Time required to rotate the root button 0 degrees during the stack closing animation.

closing_time_button_rotation is a `NumericProperty` and defaults to 0.2.

state

Indicates whether the stack is closed or open. Available options are: ‘close’, ‘open’.

state is a `OptionProperty` and defaults to ‘close’.

bg_color_root_button

Root button color in (r, g, b, a) format.

bg_color_root_button is a `ColorProperty` and defaults to [].

bg_color_stack_button

The color of the buttons in the stack (r, g, b, a) format.

bg_color_stack_button is a [ColorProperty](#) and defaults to `[]`.

color_icon_stack_button

The color icon of the buttons in the stack (r, g, b, a) format.

color_icon_stack_button is a [ColorProperty](#) and defaults to `[]`.

color_icon_root_button

The color icon of the root button (r, g, b, a) format.

color_icon_root_button is a [ColorProperty](#) and defaults to `[]`.

bg_hint_color

Background color for the text of the buttons in the stack (r, g, b, a) format.

bg_hint_color is a [ColorProperty](#) and defaults to `None`.

hint_animation

Whether to use button extension animation to display text labels.

hint_animation is a [BooleanProperty](#) and defaults to `False`.

on_open(self, *args)

Called when a stack is opened.

on_close(self, *args)

Called when a stack is closed.

on_leave(self, instance)

Called when the mouse cursor goes outside the button of stack.

on_enter(self, instance)

Called when the mouse cursor is over a button from the stack.

on_data(self, instance, value)

Creates a stack of buttons.

on_icon(self, instance, value)**on_label_text_color(self, instance, value)****on_color_icon_stack_button(self, instance, value)****on_hint_animation(self, instance, value)****on_bg_hint_color(self, instance, value)****on_color_icon_root_button(self, instance, value)****on_bg_color_stack_button(self, instance, value)****on_bg_color_root_button(self, instance, value)****set_pos_labels(self, widget)**

Sets the position of the floating labels.

set_pos_root_button(self, instance)

Sets the position of the root button.

set_pos_bottom_buttons(self, instance)

Sets the position of the bottom buttons in a stack.

open_stack(self, instance)

Opens a button stack.

```
do_animation_open_stack(self, anim_data)
close_stack(self)
    Closes the button stack.
```

2.3.29 Spinner

See also:

Material Design spec, Menus

[Circular progress indicator in Google's Material Design.](#)

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
Screen:

    MDSpinner:
        size_hint: None, None
        size: dp(46), dp(46)
        pos_hint: {'center_x': .5, 'center_y': .5}
        active: True if check.active else False

    MDCheckbox:
        id: check
        size_hint: None, None
        size: dp(48), dp(48)
        pos_hint: {'center_x': .5, 'center_y': .4}
        active: True
    ...

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```

Spinner palette

```
MDSpinner:
    # The number of color values can be any.
    palette:
        [0.28627450980392155, 0.8431372549019608, 0.596078431372549, 1],           [0.
        ↵3568627450980392, 0.3215686274509804, 0.8666666666666667, 1],           [0.
        ↵8862745098039215, 0.36470588235294116, 0.592156862745098, 1],           [0.
        ↵8784313725490196, 0.9058823529411765, 0.40784313725490196, 1],           [0.
```

```
MDSpinner(
    size_hint=(None, None),
    size=(dp(46), dp(46)),
    pos_hint={'center_x': .5, 'center_y': .5},
    active=True,
    palette=[
        [0.28627450980392155, 0.8431372549019608, 0.596078431372549, 1],
        [0.3568627450980392, 0.3215686274509804, 0.8666666666666667, 1],
        [0.8862745098039215, 0.36470588235294116, 0.592156862745098, 1],
        [0.8784313725490196, 0.9058823529411765, 0.40784313725490196, 1],
    ]
)
```

Determinate mode

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDSpinner:
        size_hint: None, None
        size: dp(48), dp(48)
        pos_hint: {'center_x': .5, 'center_y': .5}
        determinate: True
    ...

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```

API - kivymd.uix.spinner

`class kivymd.uix.spinner.MDSpinner(**kwargs)`

`MDSpinner` is an implementation of the circular progress indicator in *Google's Material Design*.

It can be used either as an indeterminate indicator that loops while the user waits for something to happen, or as a determinate indicator.

Set `determinate` to `True` to activate determinate mode, and `determinate_time` to set the duration of the animation.

Events

`on_determinate_complete` The event is called at the end of the spinner loop in the `determinate = True` mode.

`determinate`

Determinate value.

`determinate` is a `BooleanProperty` and defaults to `False`.

`determinate_time`

Determinate time value.

`determinate_time` is a `NumericProperty` and defaults to 2.

`line_width`

Progress line width of spinner.

`line_width` is a `NumericProperty` and defaults to `dp(2.25)`.

`active`

Use `active` to start or stop the spinner.

`active` is a `BooleanProperty` and defaults to `True`.

`color`

Spinner color.

`color` is a `ColorProperty` and defaults to `[0, 0, 0, 0]`.

`palette`

A set of colors. Changes with each completed spinner cycle.

`palette` is a `ListProperty` and defaults to `[]`.

`on_rotation_angle(self, *args)`

`on_palette(self, instance, value)`

`on_active(self, *args)`

`on_determinate_complete(self, *args)`

The event is called at the end of the spinner loop in the `determinate = True` mode.

`check_determinate(self, *args)`

2.3.30 Relative Layout

RelativeLayout class equivalent. Simplifies working with some widget properties. For example:

RelativeLayout

```
RelativeLayout:
    canvas:
        Color:
            rgba: app.theme_cls.primary_color
    RoundedRectangle:
        pos: (0, 0)
        size: self.size
        radius: [25, ]
```

MDRelativeLayout

```
MDRelativeLayout:
    radius: [25, ]
    md_bg_color: app.theme_cls.primary_color
```

API - kivymd.uix.relativelayout

```
class kivymd.uix.relativelayout.MDRelativeLayout(**kw)
    RelativeLayout class, see module documentation for more information.
```

2.3.31 Box Layout

BoxLayout class equivalent. Simplifies working with some widget properties. For example:

BoxLayout

```
BoxLayout:
    size_hint_y: None
    height: self.minimum_height

    canvas:
        Color:
            rgba: app.theme_cls.primary_color
    Rectangle:
        pos: self.pos
        size: self.size
```

MDBBoxLayout

```
MDBBoxLayout:  
    adaptive_height: True  
    md_bg_color: app.theme_cls.primary_color
```

Available options are:

- *adaptive_height*
- *adaptive_width*
- *adaptive_size*

adaptive_height

```
adaptive_height: True
```

Equivalent

```
size_hint_y: None  
height: self.minimum_height
```

adaptive_width

```
adaptive_width: True
```

Equivalent

```
size_hint_x: None  
height: self.minimum_width
```

adaptive_size

```
adaptive_size: True
```

Equivalent

```
size_hint: None, None  
size: self.minimum_size
```

API - kivymd.uix.boxlayout

```
class kivymd.uix.boxlayout.MDBoxLayout(**kwargs)
```

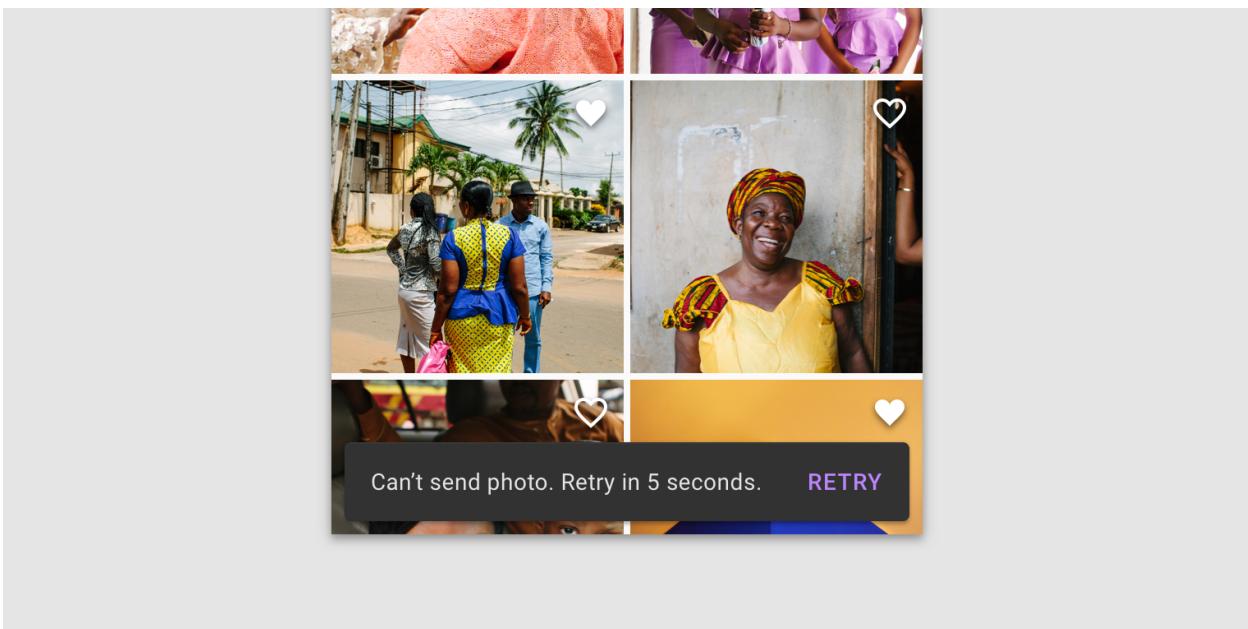
Box layout class. See module documentation for more information.

2.3.32 Snackbar

See also:

Material Design spec, Snackbars

Snackbars provide brief messages about app processes at the bottom of the screen.



Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
#:import Snackbar kivymd.uix.snackbar.Snackbar

Screen:

    MDRaisedButton:
        text: "Create simple snackbar"
        on_release: Snackbar(text="This is a snackbar!").open()
```

(continues on next page)

(continued from previous page)

```
    pos_hint: {"center_x": .5, "center_y": .5}
...
class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```

Usage with snackbar_x, snackbar_y

```
Snackbar(
    text="This is a Snackbar!",
    snackbar_x="10dp",
    snackbar_y="10dp",
    size_hint_x=(
        Window.width - (dp(10) * 2)
    ) / Window.width
).open()
```

Control width

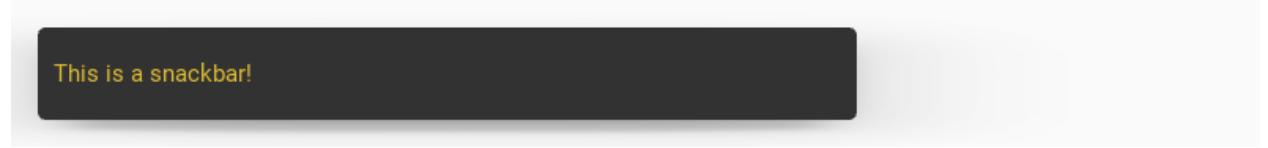
```
Snackbar(
    text="This is a Snackbar!",
    snackbar_x="10dp",
    snackbar_y="10dp",
    size_hint_x=.5
).open()
```



This is a Snackbar!

Custom text color

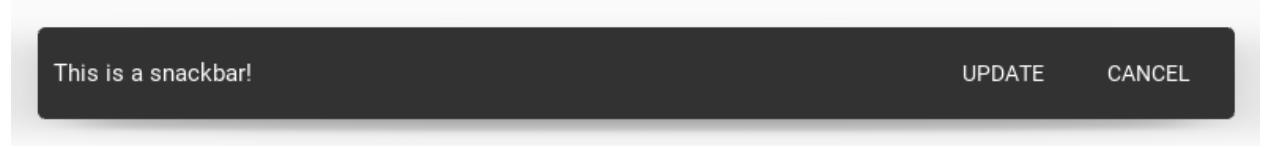
```
Snackbar(
    text="[color=#ddbb34]This is a snackbar![/color]",
    snackbar_y="10dp",
    snackbar_y="10dp",
    size_hint_x=.7
).open()
```



This is a snackbar!

Usage with button

```
snackbar = Snackbar(
    text="This is a snackbar!",
    snackbar_x="10dp",
    snackbar_y="10dp",
)
snackbar.size_hint_x = (
    Window.width - (snackbar.snackbar_x * 2)
) / Window.width
snackbar.buttons = [
    MDFlatButton(
        text="UPDATE",
        text_color=(1, 1, 1, 1),
        on_release=snackbar.dismiss,
    ),
    MDFlatButton(
        text="CANCEL",
        text_color=(1, 1, 1, 1),
        on_release=snackbar.dismiss,
    ),
]
snackbar.open()
```



This is a snackbar!

UPDATE CANCEL

Using a button with custom color

```
Snackbar(
    ...
    bg_color=(0, 0, 1, 1),
).open()
```



Custom usage

```
from kivy.lang import Builder
from kivy.animation import Animation
from kivy.clock import Clock
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.snackbar import Snackbar

KV = '''
Screen:

    MDFloatingActionButton:
        id: button
        x: root.width - self.width - dp(10)
        y: dp(10)
        on_release: app.snackbar_show()
'''

class Test(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = Builder.load_string(KV)
        self.snackbar = None
        self._interval = 0

    def build(self):
        return self.screen

    def wait_interval(self, interval):
        self._interval += interval
        if self._interval > self.snackbar.duration + 0.5:
            anim = Animation(y=dp(10), d=.2)
            anim.start(self.screen.ids.button)
            Clock.unschedule(self.wait_interval)
```

(continues on next page)

(continued from previous page)

```

    self._interval = 0
    self.snackbar = None

def snackbar_show(self):
    if not self.snackbar:
        self.snackbar = Snackbar(text="This is a snackbar!")
        self.snackbar.open()
    anim = Animation(y=dp(72), d=.2)
    anim.bind(on_complete=lambda *args: Clock.schedule_interval(
        self.wait_interval, 0))
    anim.start(self.screen.ids.button)

```

Test().run()

Custom Snackbar

```

from kivy.lang import Builder
from kivy.core.window import Window
from kivy.properties import StringProperty, NumericProperty

from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.snackbar import BaseSnackbar

KV = '''
<CustomSnackbar>

MDIconButton:
    pos_hint: {'center_y': .5}
    icon: root.icon
    opposite_colors: True

MDLabel:
    id: text_bar
    size_hint_y: None
    height: self.texture_size[1]
    text: root.text
    font_size: root.font_size
    theme_text_color: 'Custom'
    text_color: get_color_from_hex('ffffff')
    shorten: True
    shorten_from: 'right'
    pos_hint: {'center_y': .5}

```

Screen:

```
MDRaisedButton:
```

(continues on next page)

(continued from previous page)

```

        text: "SHOW"
        pos_hint: {"center_x": .5, "center_y": .45}
        on_press: app.show()
    ...

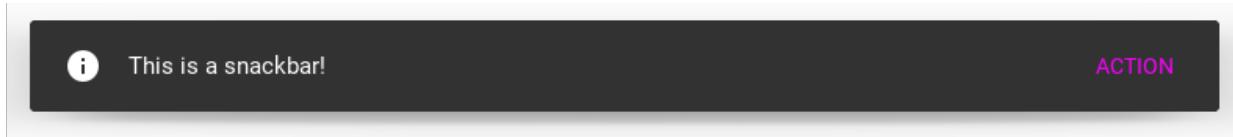
class CustomSnackbar(BaseSnackbar):
    text = StringProperty(None)
    icon = StringProperty(None)
    font_size = NumericProperty("15sp")

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

    def show(self):
        snackbar = CustomSnackbar(
            text="This is a Snackbar!",
            icon="information",
            snackbar_x="10dp",
            snackbar_y="10dp",
            buttons=[MDFlatButton(text="ACTION", text_color=(1, 1, 1, 1))]
        )
        snackbar.size_hint_x = (
            Window.width - (snackbar.snackbar_x * 2)
        ) / Window.width
        snackbar.open()

Test().run()

```



API - kivymd.uix.snackbar

```

class kivymd.uix.snackbar.Snackbar(**kwargs)
    Snackbar inherits all its functionality from BaseSnackbar

text
    The text that will appear in the snackbar.

    text is a StringProperty and defaults to ''.

font_size
    The font size of the text that will appear in the snackbar.

    font_size is a NumericProperty and defaults to '15sp'.

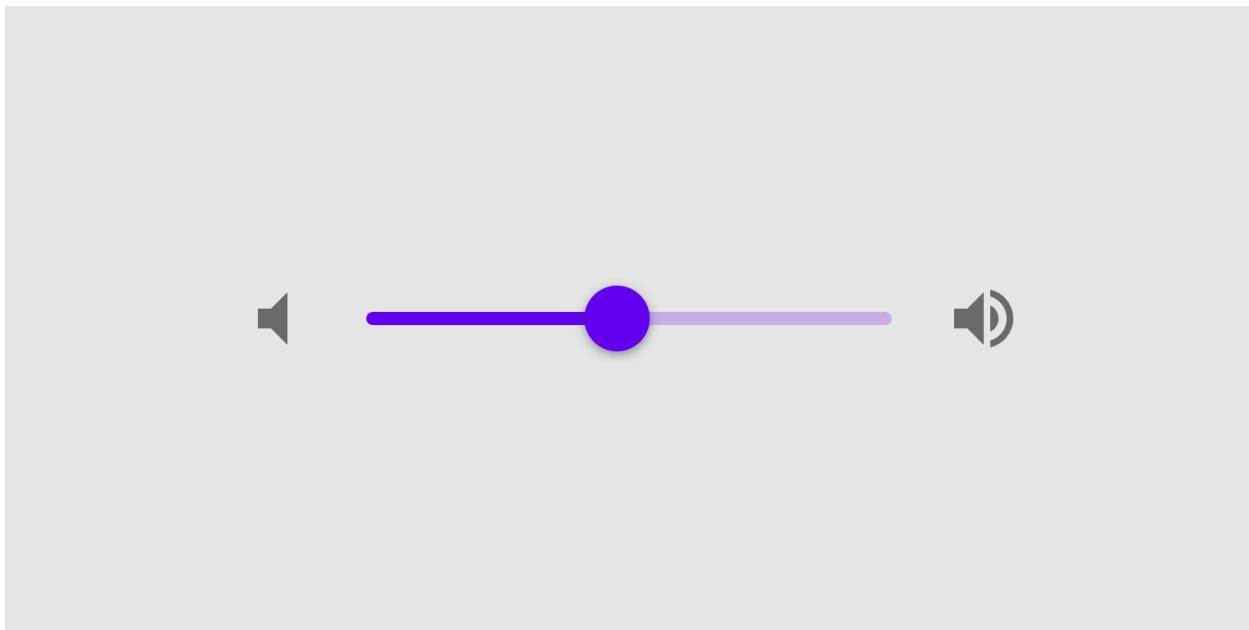
```

2.3.33 Slider

See also:

Material Design spec, Sliders

Sliders allow users to make selections from a range of values.



With value hint

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
Screen

    MDSlider:
        min: 0
        max: 100
        value: 40
    ...

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```

Without value hint

```
MDSlider:  
    min: 0  
    max: 100  
    value: 40  
    hint: False
```

Without custom color

```
MDSlider:  
    min: 0  
    max: 100  
    value: 40  
    hint: False  
    color: app.theme_cls.accent_color
```



API - kivymd.uix.slider

```
class kivymd.uix.slider.MDSlider(**kwargs)
```

Class for creating a Slider widget.

Check module documentation for more details.

active

If the slider is clicked.

`active` is an `BooleanProperty` and defaults to `False`.

hint

If True, then the current value is displayed above the slider.

`hint` is an `BooleanProperty` and defaults to `True`.

hint_bg_color

Hint rectangle color in `rgba` format.

`hint_bg_color` is an `ColorProperty` and defaults to `None`.

hint_text_color

Hint text color in `rgba` format.

hint_text_color is an `ColorProperty` and defaults to *None*.

hint_radius

Hint radius.

hint_radius is an `NumericProperty` and defaults to 4.

show_off

Show the ‘off’ ring when set to minimum value.

show_off is an `BooleanProperty` and defaults to *True*.

color

Color slider in `rgba` format.

color is an `ColorProperty` and defaults to *None*.

on_hint(self, instance, value)**on_value_normalized(self, *args)**

When the value == min set it to ‘off’ state and make slider a ring.

on_show_off(self, *args)**on_is_off(self, *args)****on_active(self, *args)****on_touch_down(self, touch)**

Receive a touch down event.

Parameters

touch: `MotionEvent` class Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

Returns bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

on_touch_up(self, touch)

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

2.3.34 Text Field

See also:

Material Design spec, Text fields

Text fields let users enter and edit text.



KivyMD provides the following field classes for use:

- [MDTextField](#)
- [MDTextFieldRound](#)
- [MDTextFieldRect](#)

Note: [MDTextField](#) inherited from [TextInput](#). Therefore, most parameters and all events of the [TextInput](#) class are also available in the [MDTextField](#) class.

MDTextField

[MDTextField](#) can be with helper text and without.

Without helper text mode

```
MDTextField:  
    hint_text: "No helper text"
```

Helper text mode on on_focus event

```
MDTextField:
    hint_text: "Helper text on focus"
    helper_text: "This will disappear when you click off"
    helper_text_mode: "on_focus"
```

Persistent helper text mode

```
MDTextField:
    hint_text: "Persistent helper text"
    helper_text: "Text is always here"
    helper_text_mode: "persistent"
```

Helper text mode ‘on_error’

To display an error in a text field when using the `helper_text_mode: "on_error"` parameter, set the “*error*” text field parameter to *True*:

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
BoxLayout:
    padding: "10dp"

    MDTextField:
        id: text_field_error
        hint_text: "Helper text on error (press 'Enter')"
        helper_text: "There will always be a mistake"
        helper_text_mode: "on_error"
        pos_hint: {"center_y": .5}
'''

class Test(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = Builder.load_string(KV)

    def build(self):
        self.screen.ids.text_field_error.bind(
            on_text_validate=self.set_error_message,
            on_focus=self.set_error_message,
        )
```

(continues on next page)

(continued from previous page)

```
return self.screen

def set_error_message(self, instance_textfield):
    self.screen.ids.text_field_error.error = True

Test().run()
```

Helper text mode ‘on_error’ (with required)

```
MDTextField:
    hint_text: "required = True"
    required: True
    helper_text_mode: "on_error"
    helper_text: "Enter text"
```

Text length control

```
MDTextField:
    hint_text: "Max text length = 5"
    max_text_length: 5
```

Multi line text

```
MDTextField:
    multiline: True
    hint_text: "Multi-line text"
```

Color mode

```
MDTextField:
    hint_text: "color_mode = 'accent'"
    color_mode: 'accent'
```

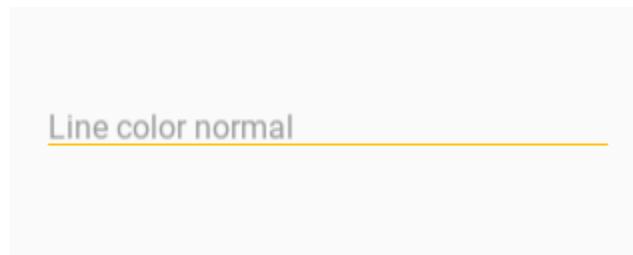
Available options are ‘primary’, ‘accent’ or ‘custom’.

MDTextField:

```
hint_text: "color_mode = 'custom'"
color_mode: 'custom'
helper_text_mode: "on_focus"
helper_text: "Color is defined by 'line_color_focus' property"
line_color_focus: 1, 0, 1, 1
```

MDTextField:

```
hint_text: "Line color normal"
line_color_normal: app.theme_cls.accent_color
```

**Rectangle mode****MDTextField:**

```
hint_text: "Rectangle mode"
mode: "rectangle"
```

Fill mode**MDTextField:**

```
hint_text: "Fill mode"
mode: "fill"
fill_color: 0, 0, 0, .4
```

Maximum height

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen

    MDTextField:
```

(continues on next page)

(continued from previous page)

```
size_hint_x: .5
hint_text: "multiline=True"
max_height: "200dp"
mode: "fill"
fill_color: 0, 0, 0, .4
multiline: True
pos_hint: {"center_x": .5, "center_y": .5}
...
class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()
```

MDTextFieldRect

Note: `MDTextFieldRect` inherited from `TextInput`. You can use all parameters and attributes of the `TextInput` class in the `MDTextFieldRect` class.

```
MDTextFieldRect:
    size_hint: 1, None
    height: "30dp"
```

Warning: While there is no way to change the color of the border.

MDTextFieldRound

Without icon

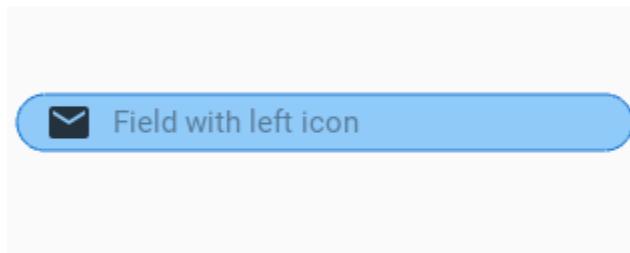
```
MDTextFieldRound:
    hint_text: 'Empty field'
```

With left icon

Warning: The icons in the `MDTextFieldRound` are static. You cannot bind events to them.

`MDTextFieldRound`:

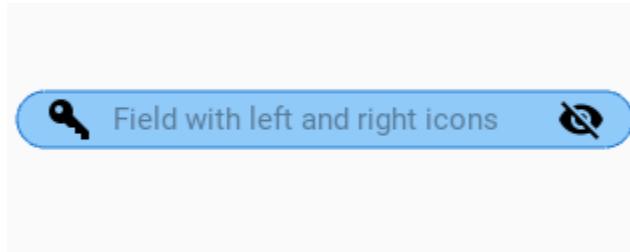
```
icon_left: "email"
hint_text: "Field with left icon"
```



With left and right icons

`MDTextFieldRound`:

```
icon_left: 'key-variant'
icon_right: 'eye-off'
hint_text: 'Field with left and right icons'
```



Control background color

`MDTextFieldRound`:

```
icon_left: 'key-variant'
normal_color: app.theme_cls.accent_color
```

`MDTextFieldRound`:

```
icon_left: 'key-variant'
normal_color: app.theme_cls.accent_color
color_active: 1, 0, 0, 1
```

Clickable icon for MDTextFieldRound

```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.relativelayout import MDRelativeLayout

KV = '''
<ClickableTextFieldRound>:
    size_hint_y: None
    height: text_field.height

    MDTextFieldRound:
        id: text_field
        hint_text: root.hint_text
        text: root.text
        password: True
        color_active: app.theme_cls.primary_light
        icon_left: "key-variant"
        padding:
            self._lbl_icon_left.texture_size[1] + dp(10) if self.icon_left else dp(15), ↴
            (self.height / 2) - (self.line_height / 2), ↴
            self._lbl_ ↴
            icon_right.texture_size[1] + dp(20), ↴
            0

        MDIconButton:
            icon: "eye-off"
            ripple_scale: .5
            pos_hint: {"center_y": .5}
            pos: text_field.width - self.width + dp(8), 0
            on_release:
                self.icon = "eye" if self.icon == "eye-off" else "eye-off"
                text_field.password = False if text_field.password is True else True

MDScreen:

    ClickableTextFieldRound:
        size_hint_x: None
        width: "300dp"
        hint_text: "Password"
        pos_hint: {"center_x": .5, "center_y": .5}
    ...

class ClickableTextFieldRound(MDRelativeLayout):
    text = StringProperty()
    hint_text = StringProperty()
    # Here specify the required parameters for MDTextFieldRound:
    # [...]
```

(continues on next page)

(continued from previous page)

```
class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```

With right icon

Note: The icon on the right is available for use in all text fields.

```
MDTextField:
    hint_text: "Name"
    mode: "fill"
    fill_color: 0, 0, 0, .4
    icon_right: "arrow-down-drop-circle-outline"
    icon_right_color: app.theme_cls.primary_color
```



```
MDTextField:
    hint_text: "Name"
    icon_right: "arrow-down-drop-circle-outline"
    icon_right_color: app.theme_cls.primary_color
```



```
MDTextField:
    hint_text: "Name"
    mode: "rectangle"
    icon_right: "arrow-down-drop-circle-outline"
    icon_right_color: app.theme_cls.primary_color
```



See also:

See more information in the [MDTextFieldRect](#) class.

API - kivymd.uix.textfield

```
class kivymd.uix.textfield.MDTextFieldRect(**kwargs)
```

TextInput class. See module documentation for more information.

Events

on_text_validate Fired only in multiline=False mode when the user hits ‘enter’. This will also unfocus the textinput.

on_double_tap Fired when a double tap happens in the text input. The default behavior selects the text around the cursor position. More info at `on_double_tap()`.

on_triple_tap Fired when a triple tap happens in the text input. The default behavior selects the line around the cursor position. More info at `on_triple_tap()`.

on_quad_touch Fired when four fingers are touching the text input. The default behavior selects the whole text. More info at `on_quad_touch()`.

Warning: When changing a TextInput property that requires re-drawing, e.g. modifying the `text`, the updates occur on the next clock cycle and not instantly. This might cause any changes to the TextInput that occur between the modification and the next cycle to be ignored, or to use previous values. For example, after a update to the `text`, changing the cursor in the same clock frame will move it using the previous text and will likely end up in an incorrect position. The solution is to schedule any updates to occur on the next clock cycle using `schedule_once()`.

Note: Selection is cancelled when TextInput is focused. If you need to show selection when TextInput is focused, you should delay (use Clock.schedule) the call to the functions for selecting text (`select_all`, `select_text`).

Changed in version 1.10.0: `background_disabled_active` has been removed.

Changed in version 1.9.0: `TextInput` now inherits from `FocusBehavior`. `keyboard_mode`, `show_keyboard()`, `hide_keyboard()`, `focus()`, and `input_type` have been removed since they are now inherited from `FocusBehavior`.

Changed in version 1.7.0: `on_double_tap`, `on_triple_tap` and `on_quad_touch` events added.

`line_anim`

If True, then text field shows animated line when on focus.

`line_anim` is an `BooleanProperty` and defaults to `True`.

```
get_rect_instruction(self)
get_color_instruction(self)
anim_rect(self, points, alpha)
```

```
class kivymd.uix.textfield.MDTextField(**kwargs)
```

TextInput class. See module documentation for more information.

Events

on_text_validate Fired only in multiline=False mode when the user hits ‘enter’. This will also unfocus the textinput.

on_double_tap Fired when a double tap happens in the text input. The default behavior selects the text around the cursor position. More info at `on_double_tap()`.

`on_triple_tap` Fired when a triple tap happens in the text input. The default behavior selects the line around the cursor position. More info at `on_triple_tap()`.

`on_quad_touch` Fired when four fingers are touching the text input. The default behavior selects the whole text. More info at `on_quad_touch()`.

Warning: When changing a `TextInput` property that requires re-drawing, e.g. modifying the `text`, the updates occur on the next clock cycle and not instantly. This might cause any changes to the `TextInput` that occur between the modification and the next cycle to be ignored, or to use previous values. For example, after a update to the `text`, changing the cursor in the same clock frame will move it using the previous text and will likely end up in an incorrect position. The solution is to schedule any updates to occur on the next clock cycle using `schedule_once()`.

Note: Selection is cancelled when `TextInput` is focused. If you need to show selection when `TextInput` is focused, you should delay (use `Clock.schedule`) the call to the functions for selecting text (`select_all`, `select_text`).

Changed in version 1.10.0: `background_disabled_active` has been removed.

Changed in version 1.9.0: `TextInput` now inherits from `FocusBehavior`. `keyboard_mode`, `show_keyboard()`, `hide_keyboard()`, `focus()`, and `input_type` have been removed since they are now inherited from `FocusBehavior`.

Changed in version 1.7.0: `on_double_tap`, `on_triple_tap` and `on_quad_touch` events added.

helper_text

Text for `helper_text` mode.

`helper_text` is an `StringProperty` and defaults to ‘*This field is required*’.

helper_text_mode

Helper text mode. Available options are: ‘`on_error`’, ‘`persistent`’, ‘`on_focus`’.

`helper_text_mode` is an `OptionProperty` and defaults to ‘`none`’.

max_text_length

Maximum allowed value of characters in a text field.

`max_text_length` is an `NumericProperty` and defaults to `None`.

required

Required text. If True then the text field requires text.

`required` is an `BooleanProperty` and defaults to `False`.

color_mode

Color text mode. Available options are: ‘`primary`’, ‘`accent`’, ‘`custom`’.

`color_mode` is an `OptionProperty` and defaults to ‘`primary`’.

mode

Text field mode. Available options are: ‘`line`’, ‘`rectangle`’, ‘`fill`’.

`mode` is an `OptionProperty` and defaults to ‘`line`’.

line_color_normal

Line color normal in `rgba` format.

`line_color_normal` is an `ColorProperty` and defaults to `None`.

line_color_focus

Line color focus in `rgba` format.

`line_color_focus` is an `ColorProperty` and defaults to `None`.

line_anim

If True, then text field shows animated line when on focus.

`line_anim` is an `BooleanProperty` and defaults to `True`.

error_color

Error color in `rgba` format for `required = True`.

`error_color` is an `ColorProperty` and defaults to `None`.

fill_color

The background color of the fill in `rgba` format when the `mode` parameter is “fill”.

`fill_color` is an `ColorProperty` and defaults to `(0, 0, 0, 0)`.

active_line

Show active line or not.

`active_line` is an `BooleanProperty` and defaults to `True`.

error

If True, then the text field goes into error mode.

`error` is an `BooleanProperty` and defaults to `False`.

current_hint_text_color

`hint_text` text color.

`current_hint_text_color` is an `ColorProperty` and defaults to `None`.

icon_right

Right icon.

`icon_right` is an `StringProperty` and defaults to ‘’.

icon_right_color

Color of right icon in `rgba` format.

`icon_right_color` is an `ColorProperty` and defaults to `(0, 0, 0, 1)`.

text_color

Text color in `rgba` format.

`text_color` is an `ColorProperty` and defaults to `None`.

font_size

Font size of the text in pixels.

`font_size` is a `NumericProperty` and defaults to `'16sp'`.

max_height

Maximum height of the text box when `multiline = True`.

`max_height` is a `NumericProperty` and defaults to `0`.

radius

The corner radius for a text field in `fill` mode.

`radius` is a `ListProperty` and defaults to `[10, 10, 0, 0]`.

font_name_helper_text
Font name for helper text.
font_name_helper_text is an `StringProperty` and defaults to ‘*Roboto*’.

font_name_hint_text
Font name for hint text.
font_name_hint_text is an `StringProperty` and defaults to ‘*Roboto*’.

font_name_max_length
Font name for max text length.
font_name_max_length is an `StringProperty` and defaults to ‘*Roboto*’.

check_text(self, interval)

set_objects_labels(self)
Creates labels objects for the parameters `helper_text`, `hint_text`, etc.

on_font_name_helper_text(self, instance, value)

on_font_name_hint_text(self, instance, value)

on_font_name_max_length(self, instance, value)

on_icon_right(self, instance, value)

on_icon_right_color(self, instance, value)

on_width(self, instance, width)
Called when the application window is resized.

on_focus(self, *args)

on_disabled(self, *args)

set_text(self, instance, text)

on_text_validate(self)

on_color_mode(self, instance, mode)

on_line_color_focus(self, *args)

on_hint_text(self, instance, value)

on_hint_text(self, instance, value)

on_height(self, instance, value)

class kivymd.uix.textfield.MDTextFieldRound(kwargs)**
TextInput class. See module documentation for more information.

Events

on_text_validate Fired only in `multiline=False` mode when the user hits ‘enter’. This will also unfocus the textinput.

on_double_tap Fired when a double tap happens in the text input. The default behavior selects the text around the cursor position. More info at `on_double_tap()`.

on_triple_tap Fired when a triple tap happens in the text input. The default behavior selects the line around the cursor position. More info at `on_triple_tap()`.

on_quad_touch Fired when four fingers are touching the text input. The default behavior selects the whole text. More info at `on_quad_touch()`.

Warning: When changing a `TextInput` property that requires re-drawing, e.g. modifying the `text`, the updates occur on the next clock cycle and not instantly. This might cause any changes to the `TextInput` that occur between the modification and the next cycle to be ignored, or to use previous values. For example, after a update to the `text`, changing the cursor in the same clock frame will move it using the previous text and will likely end up in an incorrect position. The solution is to schedule any updates to occur on the next clock cycle using `schedule_once()`.

Note: Selection is cancelled when `TextInput` is focused. If you need to show selection when `TextInput` is focused, you should delay (use `Clock.schedule`) the call to the functions for selecting text (`select_all`, `select_text`).

Changed in version 1.10.0: `background_disabled_active` has been removed.

Changed in version 1.9.0: `TextInput` now inherits from `FocusBehavior`. `keyboard_mode`, `show_keyboard()`, `hide_keyboard()`, `focus()`, and `input_type` have been removed since they are now inherited from `FocusBehavior`.

Changed in version 1.7.0: `on_double_tap`, `on_triple_tap` and `on_quad_touch` events added.

icon_left

Left icon.

`icon_left` is an `StringProperty` and defaults to ''.

icon_left_color

Color of left icon in `rgba` format.

`icon_left_color` is an `ColorProperty` and defaults to `(0, 0, 0, 1)`.

icon_right

Right icon.

`icon_right` is an `StringProperty` and defaults to ''.

icon_right_color

Color of right icon.

`icon_right_color` is an `ColorProperty` and defaults to `(0, 0, 0, 1)`.

line_color

Field line color.

`line_color` is an `ColorProperty` and defaults to `None`.

normal_color

Field color if `focus` is `False`.

`normal_color` is an `ColorProperty` and defaults to `None`.

color_active

Field color if `focus` is `True`.

`color_active` is an `ColorProperty` and defaults to `None`.

`on_focus(self, instance, value)`

`on_icon_left(self, instance, value)`

`on_icon_left_color(self, instance, value)`

`on_icon_right(self, instance, value)`

`on_icon_right_color(self, instance, value)`

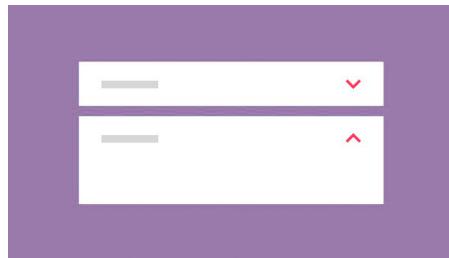
```
on_color_active(self, instance, value)
```

2.3.35 Expansion Panel

See also:

Material Design spec, Expansion panel

Expansion panels contain creation flows and allow lightweight editing of an element.



Usage

```
self.add_widget(
    MDExpansionPanel(
        icon="logo.png", # panel icon
        content=Content(), # panel content
        panel_cls=MDExpansionPanelOneLine(text="Secondary text"), # panel class
    )
)
```

To use `MDExpansionPanel` you must pass one of the following classes to the `panel_cls` parameter:

- `MDExpansionPanelOneLine`
- `MDExpansionPanelTwoLine`
- `MDExpansionPanelThreeLine`

These classes are inherited from the following classes:

- `OneLineAvatarIconListItem`
- `TwoLineAvatarIconListItem`
- `ThreeLineAvatarIconListItem`

```
self.root.ids.box.add_widget(
    MDExpansionPanel(
        icon="logo.png",
        content=Content(),
        panel_cls=MDExpansionPanelThreeLine(
            text="Text",
            secondary_text="Secondary text",
            tertiary_text="Tertiary text",
    )
)
```

(continues on next page)

(continued from previous page)

```
)  
)  
)
```

Example

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.expansionpanel import MDExpansionPanel, MDExpansionPanelThreeLine
from kivymd import images_path

KV = """
<Content>
    adaptive_height: True

    TwoLineIconListItem:
        text: "(050)-123-45-67"
        secondary_text: "Mobile"

        IconLeftWidget:
            icon: 'phone'

ScrollView:

    MDGridLayout:
        id: box
        cols: 1
        adaptive_height: True
    ...

class Content(MDBoxLayout):
    "Custom content."


class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

    def on_start(self):
        for i in range(10):
            self.root.ids.box.add_widget(
                MDExpansionPanel(
                    icon=f'{images_path}kivymd.png',
                    content=Content(),
                    panel_cls=MDExpansionPanelThreeLine(
                        text="Text",

```

(continues on next page)

(continued from previous page)

```

        secondary_text="Secondary text",
        tertiary_text="Tertiary text",
    )
)
)

Test().run()

```

Two events are available for MDExpansionPanel

- [on_open](#)
- [on_close](#)

MDExpansionPanel:

```

on_open: app.on_panel_open(args)
on_close: app.on_panel_close(args)

```

The user function takes one argument - the object of the panel:

```
def on_panel_open(self, instance_panel):
    print(instance_panel)
```

See also:

[See Expansion panel example](#)

[Expansion panel and MDCard](#)

API - kivymd.uix.expansionpanel

```
class kivymd.uix.expansionpanel.MDExpansionPanelOneLine(**kwargs)
    Single line panel.
```

```
class kivymd.uix.expansionpanel.MDExpansionPanelTwoLine(**kwargs)
    Two-line panel.
```

```
class kivymd.uix.expansionpanel.MDExpansionPanelThreeLine(**kwargs)
    Three-line panel.
```

```
class kivymd.uix.expansionpanel.MDExpansionPanelLabel(**kwargs)
    Label panel.
```

..warning:: This class is created for use in the MDStepperVertical and MDStepper classes, and has not been tested for use outside of these classes.

```
set_paddings(self, interval)
```

```
class kivymd.uix.expansionpanel.MDExpansionPanel(**kwargs)
```

Events

on_open Called when a panel is opened.

on_close Called when a panel is closed.

content

Content of panel. Must be *Kivy* widget.

content is an **ObjectProperty** and defaults to *None*.

icon

Icon of panel.

Icon Should be either be a path to an image or a logo name in *md_icons*

icon is an **StringProperty** and defaults to ‘’.

opening_transition

The name of the animation transition type to use when animating to the state ‘open’.

opening_transition is a **StringProperty** and defaults to ‘out_cubic’.

opening_time

The time taken for the panel to slide to the state ‘open’.

opening_time is a **NumericProperty** and defaults to 0.2.

closing_transition

The name of the animation transition type to use when animating to the state ‘close’.

closing_transition is a **StringProperty** and defaults to ‘out_sine’.

closing_time

The time taken for the panel to slide to the state ‘close’.

closing_time is a **NumericProperty** and defaults to 0.2.

panel_cls

Panel object. The object must be one of the classes *MDExpansionPanelOneLine*, *MDExpansionPanelTwoLine* or *MDExpansionPanelThreeLine*.

panel_cls is a **ObjectProperty** and defaults to *None*.

on_open(self, *args)

Called when a panel is opened.

on_close(self, *args)

Called when a panel is closed.

check_open_panel(self, instance)

Called when you click on the panel. Called methods to open or close a panel.

set_chevron_down(self)

Sets the chevron down.

set_chevron_up(self, instance_chevron)

Sets the chevron up.

close_panel(self, instance_panel, press_current_panel)

Method closes the panel.

open_panel(self, *args)

Method opens a panel.

get_state(self)

Returns the state of panel. Can be *close* or *open* .

`add_widget(self, widget, index=0, canvas=None)`

Add a new widget as a child of this widget.

Parameters

`widget: Widget` Widget to add to our list of children.

`index: int, defaults to 0` Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

`canvas: str, defaults to None` Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.36 Carousel

Carousel class equivalent. Simplifies working with some widget properties. For example:

Carousel

```
kv='''
YourCarousel:
    BoxLayout:
        [...]
    BoxLayout:
        [...]
    BoxLayout:
        [...]
...
builder.load_string(kv)

class YourCarousel(Carousel):
    def __init__(self, *kwargs):
        self.register_event_type("on_slide_progress")
        self.register_event_type("on_slide_complete")

    def on_touch_down(self, *args):
        ["Code to detect when the slide changes"]

    def on_touch_up(self, *args):
```

(continues on next page)

(continued from previous page)

```
["Code to detect when the slide changes"]

def calculate_slide_pos(self, *args):
    ["Code to calculate the current position of the slide"]

def do_custom_animation(self, *args)
    ["Code to recreate an animation"]
```

MDCarousel

```
MDCarousel:
    on_slide_progress:
        do_something()
    on_slide_complete:
        do_something()
```

API - kivymd.uix.carousel

class `kivymd.uix.carousel.MDCarousel(**kwargs)`
based on kivy's carousel.

See also:

`kivy.uix.carousel.Carousel`

on_slide_progress(*self*, **args*)

Event launched when the Slide animation is progress. rememebr to bind and unbid to this method.

on_slide_complete(*self*, **args*)

Event launched when the Slide animation is complete. rememebr to bind and unbid to this method.

on_touch_down(*self*, *touch*)

Receive a touch down event.

Parameters

touch: MotionEvent class Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

Returns bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

on_touch_up(*self*, *touch*)

Receive a touch up event. The touch is in parent coordinates.

See `on_touch_down()` for more information.

2.3.37 StackLayout

StackLayout class equivalent. Simplifies working with some widget properties. For example:

StackLayout

```
StackLayout:
    size_hint_y: None
    height: self.minimum_height

    canvas:
        Color:
            rgba: app.theme_cls.primary_color
        Rectangle:
            pos: self.pos
            size: self.size
```

MDStackLayout

```
MDStackLayout:
    adaptive_height: True
    md_bg_color: app.theme_cls.primary_color
```

Available options are:

- *adaptive_height*
- *adaptive_width*
- *adaptive_size*

adaptive_height

```
adaptive_height: True
```

Equivalent

```
size_hint_y: None
height: self.minimum_height
```

adaptive_width

```
adaptive_width: True
```

Equivalent

```
size_hint_x: None  
width: self.minimum_width
```

adaptive_size

```
adaptive_size: True
```

Equivalent

```
size_hint: None, None  
size: self.minimum_size
```

API - kivymd.uix.stacklayout

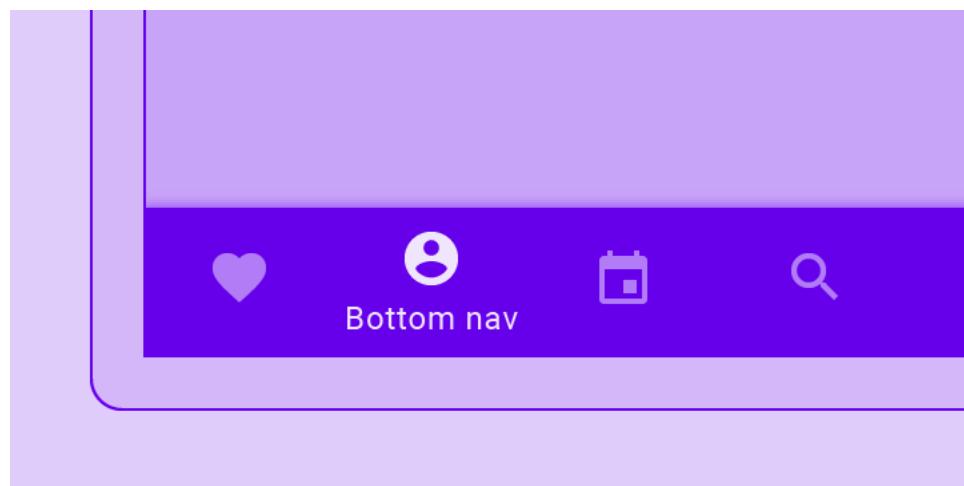
```
class kivymd.uix.stacklayout.MDStackLayout(**kwargs)  
Stack layout class. See module documentation for more information.
```

2.3.38 Bottom Navigation

See also:

Material Design spec, Bottom navigation

Bottom navigation bars allow movement between primary destinations in an app:



Usage

```
<Root>>:
    MDBottomNavigation:
        MDBottomNavigationItem:
            name: "screen 1"
            YourContent:
        MDBottomNavigationItem:
            name: "screen 2"
            YourContent:
        MDBottomNavigationItem:
            name: "screen 3"
            YourContent:
```

For ease of understanding, this code works like this:

```
<Root>>:
    ScreenManager:
        Screen:
            name: "screen 1"
            YourContent:
        Screen:
            name: "screen 2"
            YourContent:
        Screen:
            name: "screen 3"
            YourContent:
```

Example

```
from kivymd.app import MDApp
from kivy.lang import Builder

class Test(MDApp):
    def build(self):
```

(continues on next page)

(continued from previous page)

```
self.theme_cls.primary_palette = "Gray"
return Builder.load_string(
    """
BoxLayout:
    orientation:'vertical'

    MDToolbar:
        title: 'Bottom navigation'
        md_bg_color: .2, .2, .2, 1
        specific_text_color: 1, 1, 1, 1

    MDBottomNavigation:
        panel_color: .2, .2, .2, 1

        MDBottomNavigationItem:
            name: 'screen 1'
            text: 'Python'
            icon: 'language-python'

            MDLabel:
                text: 'Python'
                halign: 'center'

        MDBottomNavigationItem:
            name: 'screen 2'
            text: 'C++'
            icon: 'language-cpp'

            MDLabel:
                text: 'I programming of C++'
                halign: 'center'

        MDBottomNavigationItem:
            name: 'screen 3'
            text: 'JS'
            icon: 'language-javascript'

            MDLabel:
                text: 'JS'
                halign: 'center'
    """

)
Test().run()
```

`MDBottomNavigationItem` provides the following events for use:

```
__events__ = (
    "on_tab_touch_down",
    "on_tab_touch_move",
    "on_tab_touch_up",
    "on_tab_press",
    "on_tab_release",
)
```

See also:

See `__events__`

Root:

`MDBottomNavigation:`

```
MDBottomNavigationItem:
    on_tab_touch_down: print("on_tab_touch_down")
    on_tab_touch_move: print("on_tab_touch_move")
    on_tab_touch_up: print("on_tab_touch_up")
    on_tab_press: print("on_tab_press")
    on_tab_release: print("on_tab_release")
```

`YourContent:`

How to automatically switch a tab?

Use method `switch_tab` which takes as argument the name of the tab you want to switch to.

How to change icon color?

`MDBottomNavigation:`

```
text_color_active: 1, 0, 1, 1
```



C++

JS

`MDBottomNavigation:`

```
text_color_normal: 1, 0, 1, 1
```



C++

JS

See also:

See Tab auto switch example

See full example

API - kivymd.uix.bottomnavigation

```
class kivymd.uix.bottomnavigation.MDTab(**kwargs)
```

A tab is simply a screen with meta information that defines the content that goes in the tab header.

text

Tab header text.

`text` is an `StringProperty` and defaults to ''.

icon

Tab header icon.

`icon` is an `StringProperty` and defaults to '`checkbox-blank-circle`'.

```
on_tab_touch_down(self, *args)
```

```
on_tab_touch_move(self, *args)
```

```
on_tab_touch_up(self, *args)
```

```
on_tab_press(self, *args)
```

```
on_tab_release(self, *args)
```

```
class kivymd.uix.bottomnavigation.MDBottomNavigationItem(**kwargs)
```

A tab is simply a screen with meta information that defines the content that goes in the tab header.

header

`header` is an `MDBottomNavigationHeader` and defaults to `None`.

```
on_tab_press(self, *args)
```

```
on_leave(self, *args)
```

```
class kivymd.uix.bottomnavigation.TabbedPanelBase(**kwargs)
```

A class that contains all variables a TabPannel must have. It is here so I (zingballyhoo) don't get mad about the TabbedPannels not being DRY.

current

Current tab name.

`current` is an `StringProperty` and defaults to `None`.

previous_tab

`previous_tab` is an `MDTab` and defaults to `None`.

panel_color

Panel color of bottom navigation.

`panel_color` is an `ListProperty` and defaults to `[]`.

tabs

```
class kivymd.uix.bottomnavigation.MDBottomNavigation(**kwargs)
```

A bottom navigation that is implemented by delegating all items to a ScreenManager.

first_widget

`first_widget` is an `MDBottomNavigationItem` and defaults to `None`.

tab_header

`tab_header` is an `MDBottomNavigationHeader` and defaults to `None`.

text_color_normal

Text color of the label when it is not selected.

`text_color_normal` is an `ListProperty` and defaults to `[1, 1, 1, 1]`.

`text_color_active`

Text color of the label when it is selected.

`text_color_active` is an `ListProperty` and defaults to `[1, 1, 1, 1]`.

`on_panel_color(self, instance, value)`

`on_text_color_normal(self, instance, value)`

`on_text_color_active(self, instance, value)`

`switch_tab(self, name_tab)`

Switching the tab by name.

`refresh_tabs(self)`

Refresh all tabs.

`on_size(self, *args)`

`on_resize(self, instance=None, width=None, do_again=True)`

Called when the application window is resized.

`add_widget(self, widget, **kwargs)`

Add a new widget as a child of this widget.

Parameters

`widget: Widget` Widget to add to our list of children.

`index: int, defaults to 0` Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

`canvas: str, defaults to None` Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

`remove_widget(self, widget)`

Remove a widget from the children of this widget.

Parameters

`widget: Widget` Widget to remove from our children list.

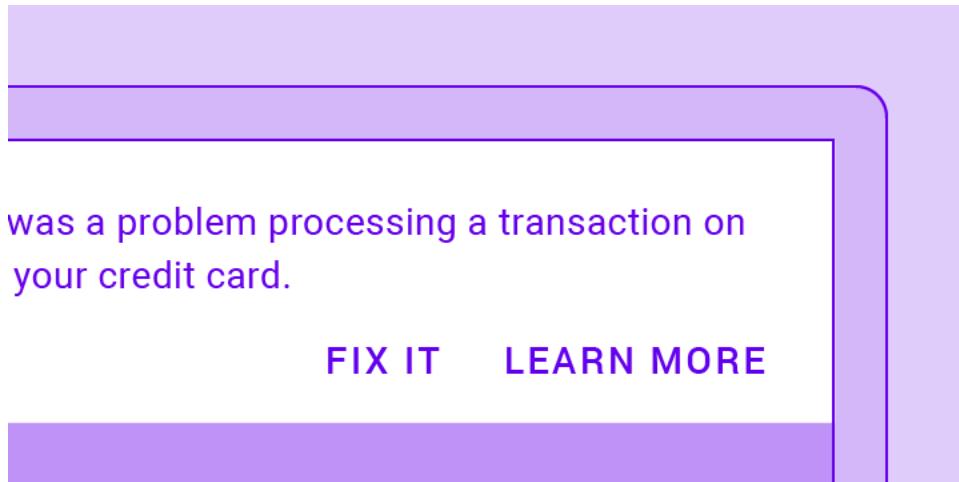
```
>>> from kivy.uix.button import Button
>>> root = Widget()
>>> button = Button()
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

2.3.39 Banner

See also:

Material Design spec, Banner

A banner displays a prominent message and related optional actions.



Usage

```
from kivy.lang import Builder
from kivy.factory import Factory

from kivymd.app import MDApp

Builder.load_string('''
<ExampleBanner@Screen>

MDBanner:
    id: banner
    text: ["One line string text example without actions."]
    # The widget that is under the banner.
    # It will be shifted down to the height of the banner.
    over_widget: screen
    vertical_pad: toolbar.height

MDToolbar:
    id: toolbar
    title: "Example Banners"
    elevation: 10
    pos_hint: {'top': 1}

BoxLayout:
    id: screen
```

(continues on next page)

(continued from previous page)

```

        orientation: "vertical"
        size_hint_y: None
        height: Window.height - toolbar.height

    OneLineListItem:
        text: "Banner without actions"
        on_release: banner.show()

    Widget:
''')

class Test(MDApp):
    def build(self):
        return Factory.ExampleBanner()

Test().run()

```

Banner type.

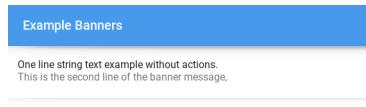
By default, the banner is of the type 'one-line':

```
MDBanner:
    text: ["One line string text example without actions."]
```



To use a two-line banner, specify the 'two-line' `MDBanner.type` for the banner and pass the list of two lines to the `MDBanner.text` parameter:

```
MDBanner:
    type: "two-line"
    text:
        ["One line string text example without actions.", "This is the second line of
         ↵the banner message."]
```



Similarly, create a three-line banner:

```
MDBanner:
    type: "three-line"
    text:
        ["One line string text example without actions.", "This is the second line of
         ↵the banner message.", "and this is the third line of the banner message."]
```

Example Banners

One line string text example without actions.
This is the second line of the banner message,
and this is the third line of the banner message.

To add buttons to any type of banner, use the `MDBanner.left_action` and `MDBanner.right_action` parameters, which should take a list ['Button name', function]:

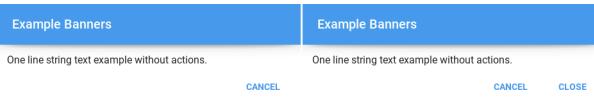
`MDBanner:`

```
text: ["One line string text example without actions."]
left_action: ["CANCEL", lambda x: None]
```

Or two buttons:

`MDBanner:`

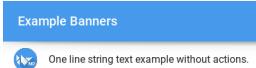
```
text: ["One line string text example without actions."]
left_action: ["CANCEL", lambda x: None]
right_action: ["CLOSE", lambda x: None]
```



If you want to use the icon on the left in the banner, add the prefix '`-icon`' to the banner type:

`MDBanner:`

```
type: "one-line-icon"
icon: f'{images_path}/kivymd.png'
text: ["One line string text example without actions."]
```



Note: See full example

API - kivymd.uix.banner

`class kivymd.uix.banner.MDBanner(**kwargs)`

FakeRectangularElevationBehavior is a shadow mockup for widgets. Improves performance using cached images inside `kivymd.images` dir

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
    ThemableBehavior,
    FakeCircularElevationBehavior,
    SpecificBackgroundColorBehavior,
    # here you add the other front end classes for the widget front_end,
```

(continues on next page)

(continued from previous page)

```
):  
[...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors. *FakeCircularElevationBehavior* will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(  
    MDCard,  
    FakeRectangularElevationBehavior  
):  
[...]
```

Note: About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

vertical_pad

Indent the banner at the top of the screen.

vertical_pad is an [NumericProperty](#) and defaults to *dp(68)*.

opening_transition

The name of the animation transition.

opening_transition is an [StringProperty](#) and defaults to '*in_quad*'.

icon

Icon banner.

icon is an [StringProperty](#) and defaults to '*data/logo/kivy-icon-128.png*'.

over_widget

The widget that is under the banner. It will be shifted down to the height of the banner.

over_widget is an [ObjectProperty](#) and defaults to *None*.

text

List of lines for banner text. Must contain no more than three lines for a '*one-line*', '*two-line*' and '*three-line*' banner, respectively.

text is an [ListProperty](#) and defaults to *[]*.

left_action

The action of banner.

To add one action, make a list [*'name_action'*, *callback*] where '*name_action*' is a string that corresponds to an action name and *callback* is the function called on a touch release event.

left_action is an [ListProperty](#) and defaults to *[]*.

right_action

Works the same way as *left_action*.

right_action is an [ListProperty](#) and defaults to *[]*.

type

Banner type. . Available options are: (“one-line”, “two-line”, “three-line”, “one-line-icon”, “two-line-icon”, “three-line-icon”).

`type` is an [OptionProperty](#) and defaults to ‘one-line’.

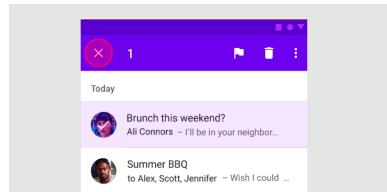
`add_actions_buttons(self, box, data)`
`set_left_action(self)`
`set_right_action(self)`
`set_type_banner(self)`
`add_banner_to_container(self)`
`show(self)`
`animation_display_banner(self, i)`
`hide(self)`

2.3.40 Selection

See also:

[Material Design spec, Banner](#)

Selection refers to how users indicate specific items they intend to take action on.



Entering selection mode

To select an item and enter selection mode, long press the item:

Exiting selection mode

To exit selection mode, tap each selected item until they’re all deselected:

Larger selections

Note: This feature is missing yet.

Events

```
def on_selected(self, instance_selection_list, instance_selection_item):
    "Called when a list item is selected."
    pass

def on_unselected(self, instance_selection_list, instance_selection_item):
    "Called when a list item is unselected."
    pass
```

Example with TwoLineAvatarListItem

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.utils import get_color_from_hex

from kivymd.app import MDApp
from kivymd.uix.list import TwoLineAvatarListItem

KV = '''
<MyItem>
    text: "Two-line item with avatar"
    secondary_text: "Secondary text here"
    _no_ripple_effect: True

    ImageLeftWidget:
        source: "data/logo/kivy-icon-256.png"

MDBBoxLayout:
    orientation: "vertical"

    MDToolbar:
        id: toolbar
        title: "Inbox"
        left_action_items: [["menu"]]
        right_action_items: [["magnify"], ["dots-vertical"]]
        md_bg_color: 0, 0, 0, 1

    MDBBoxLayout:
        padding: "24dp", "8dp", 0, "8dp"
        adaptive_size: True

        MDLabel:
            text: "Today"
            adaptive_size: True
```

(continues on next page)

(continued from previous page)

```

ScrollView:

    MDSelectionList:
        id: selection_list
        spacing: "12dp"
        overlay_color: app.overlay_color[:-1] + [.2]
        icon_bg_color: app.overlay_color
        on_selected: app.on_selected(*args)
        on_unselected: app.on_unselected(*args)
        on_selected_mode: app.set_selection_mode(*args)
    ...

class MyItem(TwoLineAvatarListItem):
    pass


class Example(MDApp):
    overlay_color = get_color_from_hex("#6042e4")

    def build(self):
        return Builder.load_string(KV)

    def on_start(self):
        for i in range(10):
            self.root.ids.selection_list.add_widget(MyItem())

    def set_selection_mode(self, instance_selection_list, mode):
        if mode:
            md_bg_color = self.overlay_color
            left_action_items = [
                [
                    "close",
                    lambda x: self.root.ids.selection_list.unselected_all(),
                ]
            ]
            right_action_items = [["trash-can"], ["dots-vertical"]]
        else:
            md_bg_color = (0, 0, 0, 1)
            left_action_items = [["menu"]]
            right_action_items = [["magnify"], ["dots-vertical"]]
            self.root.ids.toolbar.title = "Inbox"

        Animation(md_bg_color=md_bg_color, d=0.2).start(self.root.ids.toolbar)
        self.root.ids.toolbar.left_action_items = left_action_items
        self.root.ids.toolbar.right_action_items = right_action_items

    def on_selected(self, instance_selection_list, instance_selection_item):
        self.root.ids.toolbar.title = str(
            len(instance_selection_list.get_selected_list_items())
    )

```

(continues on next page)

(continued from previous page)

```
def on_unselected(self, instance_selection_list, instance_selection_item):
    if instance_selection_list.get_selected_list_items():
        self.root.ids.toolbar.title = str(
            len(instance_selection_list.get_selected_list_items())
    )
```

Example().run()

Example with FitImage

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.utils import get_color_from_hex

from kivymd.app import MDApp
from kivymd.utils.fitimage import FitImage

KV = '''
MDBoxLayout:
    orientation: "vertical"
    md_bg_color: app.theme_cls.bg_light

    MDToolbar:
        id: toolbar
        title: "Inbox"
        left_action_items: [["menu"]]
        right_action_items: [["magnify"], ["dots-vertical"]]
        md_bg_color: app.theme_cls.bg_light
        specific_text_color: 0, 0, 0, 1

    MDBoxLayout:
        padding: "24dp", "8dp", 0, "8dp"
        adaptive_size: True

        MDLabel:
            text: "Today"
            adaptive_size: True

    ScrollView:

        MDSelectionList:
            id: selection_list
            padding: "24dp", 0, "24dp", "24dp"
            cols: 3
            spacing: "12dp"
            overlay_color: app.overlay_color[:-1] + [.2]
            icon_bg_color: app.overlay_color
```

(continues on next page)

(continued from previous page)

```

progress_round_color: app.progress_round_color
on_selected: app.on_selected(*args)
on_unselected: app.on_unselected(*args)
on_selected_mode: app.set_selection_mode(*args)
...
```
class Example(MDApp):
 overlay_color = get_color_from_hex("#6042e4")
 progress_round_color = get_color_from_hex("#ef514b")

 def build(self):
 return Builder.load_string(KV)

 def on_start(self):
 for i in range(10):
 self.root.ids.selection_list.add_widget(
 FitImage(
 source="image.png",
 size_hint_y=None,
 height="240dp",
)
)

 def set_selection_mode(self, instance_selection_list, mode):
 if mode:
 md_bg_color = self.overlay_color
 left_action_items = [
 [
 "close",
 lambda x: self.root.ids.selection_list.unselected_all(),
]
]
 right_action_items = [["trash-can"], ["dots-vertical"]]
 else:
 md_bg_color = (1, 1, 1, 1)
 left_action_items = [["menu"]]
 right_action_items = [["magnify"], ["dots-vertical"]]
 self.root.ids.toolbar.title = "Inbox"

 Animation(md_bg_color=md_bg_color, d=0.2).start(self.root.ids.toolbar)
 self.root.ids.toolbar.left_action_items = left_action_items
 self.root.ids.toolbar.right_action_items = right_action_items

 def on_selected(self, instance_selection_list, instance_selection_item):
 self.root.ids.toolbar.title = str(
 len(instance_selection_list.get_selected_list_items())
)

 def on_unselected(self, instance_selection_list, instance_selection_item):
 if instance_selection_list.get_selected_list_items():
 self.root.ids.toolbar.title = str(
```

```

(continues on next page)

(continued from previous page)

```
    len(instance_selection_list.get_selected_list_items())
)
```

```
Example().run()
```

API - kivymd.uix.selection

```
class kivymd.uix.selection.MDSelectionList(**kwargs)
```

Events

on_selected Called when a list item is selected.

on_unselected Called when a list item is unselected.

selected_mode

List item selection mode. If *True* when clicking on a list item, it will be selected.

selected_mode is an [BooleanProperty](#) and defaults to *False*.

icon

Name of the icon with which the selected list item will be marked.

icon is an [StringProperty](#) and defaults to ‘check’.

icon_pos

The position of the icon that will mark the selected list item.

icon_pos is an [ListProperty](#) and defaults to *[]*.

icon_bg_color

Background color of the icon that will mark the selected list item.

icon_bg_color is an [ColorProperty](#) and defaults to *[1, 1, 1, 1]*.

icon_check_color

Color of the icon that will mark the selected list item.

icon_check_color is an [ColorProperty](#) and defaults to *[1, 1, 1, 1]*.

overlay_color

The overlay color of the selected list item..

overlay_color is an [ColorProperty](#) and defaults to *[0, 0, 0, 0.2]*.

progress_round_size

Size of the spinner for switching of *selected_mode* mode.

progress_round_size is an [NumericProperty](#) and defaults to *dp(46)*.

progress_round_color

Color of the spinner for switching of *selected_mode* mode.

progress_round_color is an [NumericProperty](#) and defaults to *None*.

add_widget(self, widget, index=0, canvas=None)

Add a new widget as a child of this widget.

Parameters

widget: Widget Widget to add to our list of children.

index: int, defaults to 0 Index to insert the widget in the list. Notice that the default of 0 means the widget is inserted at the beginning of the list and will thus be drawn on top of other sibling widgets. For a full discussion of the index and widget hierarchy, please see the [Widgets Programming Guide](#).

New in version 1.0.5.

canvas: str, defaults to None Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

New in version 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

get_selected(self)

get_selected_list_items(self)

unselected_all(self)

selected_all(self)

on_selected(self, *args)

Called when a list item is selected.

on_unselected(self, *args)

Called when a list item is unselected.

2.4 Behaviors

2.4.1 Hover

Changing when the mouse is on the widget and the widget is visible.

To apply hover behavior, you must create a new class that is inherited from the widget to which you apply the behavior and from the `HoverBehavior` class.

In *KV file*:

```
<HoverItem@MDBoxLayout+ThemableBehavior+HoverBehavior>
```

In *python file*:

```
class HoverItem(MDBoxLayout, ThemableBehavior, HoverBehavior):
    "Custom item implementing hover behavior."
```

After creating a class, you must define two methods for it: `HoverBehavior.on_enter` and `HoverBehavior.on_leave`, which will be automatically called when the mouse cursor is over the widget and when the mouse cursor goes beyond the widget.

Note: `HoverBehavior` will by default check to see if the current Widget is visible (i.e. not covered by a modal or popup and not a part of a Relative Layout, MDTab or Carousel that is not currently visible etc) and will only issue events if the widget is visible.

To get the legacy behavior that the events are always triggered, you can set `detect_visible` on the Widget to `False`.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import HoverBehavior
from kivymd.uix.boxlayout import MDBBoxLayout
from kivymd.theming import ThemableBehavior

KV = '''
Screen

    MDBBoxLayout:
        id: box
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: .8, .8
        md_bg_color: app.theme_cls.bg_darkest
'''


class HoverItem(MDBBoxLayout, ThemableBehavior, HoverBehavior):
    "Custom item implementing hover behavior."

    def on_enter(self, *args):
        "The method will be called when the mouse cursor
        is within the borders of the current widget."
        self.md_bg_color = (1, 1, 1, 1)

    def on_leave(self, *args):
        "The method will be called when the mouse cursor goes beyond
        the borders of the current widget."
        self.md_bg_color = self.theme_cls.bg_darkest


class Test(MDApp):
    def build(self):
        self.screen = Builder.load_string(KV)
        for i in range(5):
            self.screen.ids.box.add_widget(HoverItem())
        return self.screen


Test().run()
```

API - kivymd.uix.behaviors.hover_behavior

```
class kivymd.uix.behaviors.hover_behavior.HoverBehavior(**kwargs)
```

Events

on_enter Called when mouse enters the bbox of the widget AND the widget is visible

on_leave Called when the mouse exits the widget AND the widget is visible

hovering

True, if the mouse cursor is within the borders of the widget.

Note that this is set and cleared even if the widget is not visible

`hover` is a `BooleanProperty` and defaults to `False`.

hover_visible

True if hovering is `True` AND is the current widget is visible

`hover_visible` is a `BooleanProperty` and defaults to `False`.

enter_point

Holds the last position where the mouse pointer crossed into the Widget if the Widget is visible and is currently in a hovering state

`enter_point` is a `ObjectProperty` and defaults to `None`.

detect_visible

Should this widget perform the visibility check?

`detect_visible` is a `BooleanProperty` and defaults to `True`.

on_mouse_update(self, *args)

on_enter(self)

Called when mouse enters the bbox of the widget AND the widget is visible.

on_leave(self)

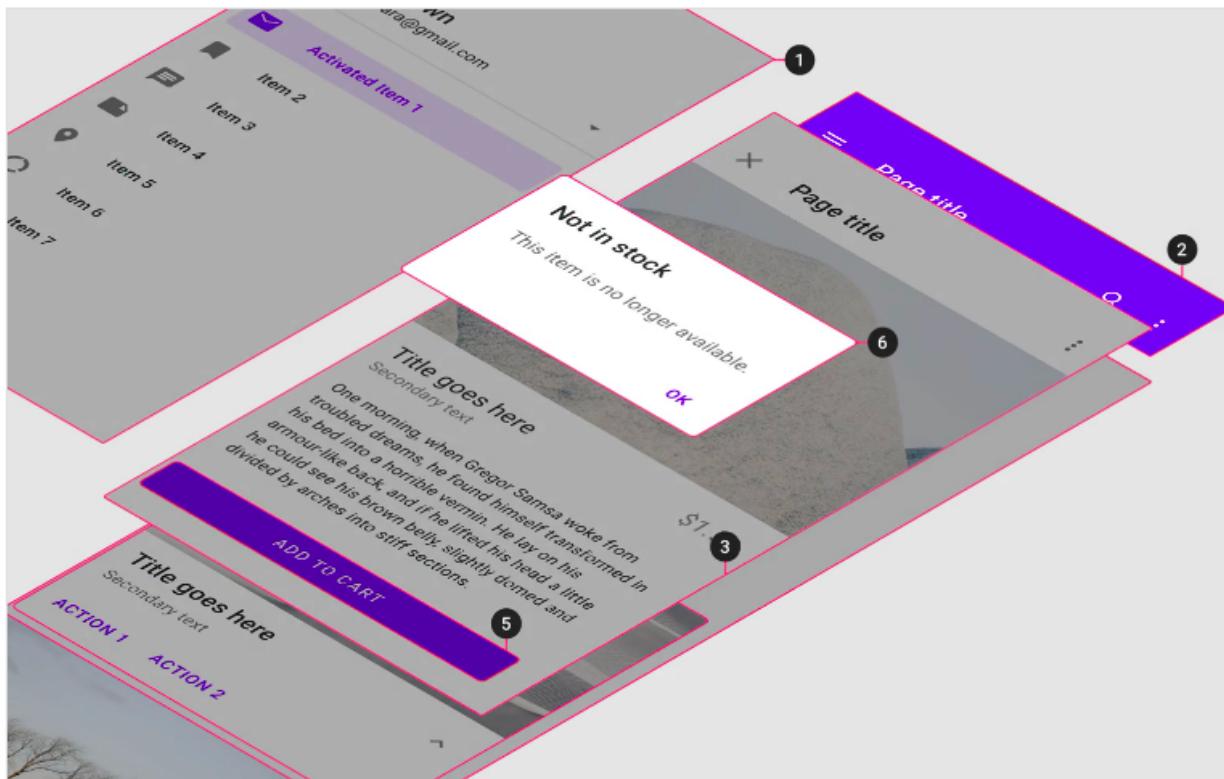
Called when the mouse exits the widget AND the widget is visible.

2.4.2 Elevation

See also:

Material Design spec, Elevation

Elevation is the relative distance between two surfaces along the z-axis.



There are 5 classes in KivyMD that can simulate shadow:

1. `FakeRectangularElevationBehavior`
2. `FakeCircularElevationBehavior`
3. `RectangularElevationBehavior`
4. `CircularElevationBehavior`
5. `RoundedRectangularElevationBehavior`

By default, KivyMD widgets use the elevation behavior implemented in classes `FakeRectangularElevationBehavior` and `FakeCircularElevationBehavior` for cast shadows. These classes use the old method of rendering shadows and it doesn't look very aesthetically pleasing. Shadows are harsh, no softness:

The `RectangularElevationBehavior`, `CircularElevationBehavior`, `RoundedRectangularElevationBehavior` classes use the new shadow rendering algorithm, based on textures creation using the `Pillow` library. It looks very aesthetically pleasing and beautiful.

Warning: Remember that `RectangularElevationBehavior`, `CircularElevationBehavior`, `RoundedRectangularElevationBehavior` classes require a lot of resources from the device on which your application will run, so you should not use these classes on mobile devices.

```
from kivy.lang import Builder
from kivy.uix.widget import Widget
```

(continues on next page)

(continued from previous page)

```
from kivymd.app import MDApp
from kivymd.uix.card import MDCard
from kivymd.uix.behaviors import RectangularElevationBehavior
from kivymd.uix.boxlayout import MDBoxLayout

KV = """
<Box@MDBBoxLayout>
    adaptive_size: True
    orientation: "vertical"
    spacing: "36dp"

<BaseShadowWidget>
    size_hint: None, None
    size: 100, 100
    md_bg_color: 0, 0, 1, 1
    elevation: 36
    pos_hint: {'center_x': .5}

MDFloatLayout:

    MDBBoxLayout:
        adaptive_size: True
        pos_hint: {'center_x': .5, 'center_y': .5}
        spacing: "56dp"

    Box:

        MDLabel:
            text: "Deprecated shadow rendering"
            adaptive_size: True

        DeprecatedShadowWidget:

            MDLabel:
                text: "Doesn't require a lot of resources"
                adaptive_size: True

        Box:

            MDLabel:
                text: "New shadow rendering"
                adaptive_size: True

        NewShadowWidget:

            MDLabel:
                text: "It takes a lot of resources"
                adaptive_size: True
..."
```

(continues on next page)

(continued from previous page)

```

class BaseShadowWidget(Widget):
    pass

class DeprecatedShadowWidget(MDCard, BaseShadowWidget):
    "Deprecated shadow rendering. Doesn't require a lot of resources."

class NewShadowWidget(RectangularElevationBehavior, BaseShadowWidget, MDBoxLayout):
    "New shadow rendering. It takes a lot of resources."

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()

```

Deprecated shadow rendering



Doesn't require a lot of resources

New shadow rendering



It takes a lot of resources

For example, let's create an button with a rectangular elevation effect:

```

from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import (
    RectangularRippleBehavior,
    BackgroundColorBehavior,
    FakeRectangularElevationBehavior,
)
KV = '''
<RectangularElevationButton>:
    size_hint: None, None

```

(continues on next page)

(continued from previous page)

```

size: "250dp", "50dp"

MDScreen:

    # With elevation effect
    RectangularElevationButton:
        pos_hint: {"center_x": .5, "center_y": .6}
        elevation: 18

    # Without elevation effect
    RectangularElevationButton:
        pos_hint: {"center_x": .5, "center_y": .4}
    ...

class RectangularElevationButton(
    RectangularRippleBehavior,
    FakeRectangularElevationBehavior,
    ButtonBehavior,
    BackgroundColorBehavior,
):
    md_bg_color = [0, 0, 1, 1]

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()

```

Similarly, create a circular button:

```

from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.app import MDApp
from kivymd.uix.behaviors import (
    CircularRippleBehavior,
    FakeCircularElevationBehavior,
)
KV = '''
<CircularElevationButton>:
    size_hint: None, None
    size: "100dp", "100dp"
    radius: self.size[0] / 2
    md_bg_color: 0, 0, 1, 1

```

(continues on next page)

(continued from previous page)

```
MDIcon:
    icon: "hand-heart"
    halign: "center"
    valign: "center"
    size: root.size
    pos: root.pos
    font_size: root.size[0] * .6
    theme_text_color: "Custom"
    text_color: [1] * 4

MDScreen:

    CircularElevationButton:
        pos_hint: {"center_x": .5, "center_y": .6}
        elevation: 24
    ...

class CircularElevationButton(
    FakeCircularElevationBehavior,
    CircularRippleBehavior,
    ButtonBehavior,
    MDBBoxLayout,
):
    pass

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()
```



Animating the elevation

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.properties import ObjectProperty
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.theming import ThemableBehavior
from kivymd.uix.behaviors import FakeRectangularElevationBehavior,
    RectangularRippleBehavior
from kivymd.uix.boxlayout import MDBBoxLayout

KV = '''
MDFloatLayout:

    ElevatedWidget:
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: None, None
        size: 100, 100
        md_bg_color: 0, 0, 1, 1
'''


class ElevatedWidget(ThemableBehavior,
    FakeRectangularElevationBehavior,
    RectangularRippleBehavior,
    ButtonBehavior,
    MDBBoxLayout,
):
    shadow_animation = ObjectProperty()

    def on_press(self, *args):
        if self.shadow_animation:
            Animation.cancel_all(self, "_elevation")
        self.shadow_animation = Animation(_elevation=self.elevation + 10, d=0.4)
        self.shadow_animation.start(self)

    def on_release(self, *args):
        if self.shadow_animation:
            Animation.cancel_all(self, "_elevation")
        self.shadow_animation = Animation(_elevation=self.elevation, d=0.1)
        self.shadow_animation.start(self)

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()
```

Lighting position

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.card import MDCard
from kivymd.uix.boxlayout import MDBBoxLayout
from kivymd.uix.behaviors import RectangularElevationBehavior

KV = """
MDScreen:

    ShadowCard:
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: None, None
        size: 100, 100
        shadow_pos: -10 + slider.value, -10 + slider.value
        elevation: 24
        md_bg_color: 1, 1, 1, 1

    MDSlider:
        id: slider
        max: 20
        size_hint_x: .6
        pos_hint: {'center_x': .5, 'center_y': .3}
    ...

class ShadowCard(RectangularElevationBehavior, MDBBoxLayout):
    pass

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()
```

API - kivymd.uix.behaviors.elevation

```
class kivymd.uix.behaviors.elevation.CommonElevationBehavior(**kwargs)
    Common base class for rectangular and circular elevation behavior.
```

elevation

Elevation of the widget.

Note: Although, this value does not represent the current elevation of the widget. `_elevation` can be used to animate the current elevation and come back using the `elevation` property directly.

For example:

```
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import CircularElevationBehavior,
    CircularRippleBehavior
from kivymd.uix.boxlayout import MDBBoxLayout

KV = '''
#:import Animation kivy.animation.Animation


<WidgetWithShadow>
    size_hint: [None, None]
    elevation: 6
    animation_: None
    md_bg_color: [1] * 4
    on_size:
        self.radius = [self.height / 2] * 4
    on_press:
        if self.animation_:
            self.animation_.cancel(self);
        self.animation_ = Animation(_elevation=self.elevation + 6,
            d=0.08);
        self.animation_.start(self)
    on_release:
        if self.animation_:
            self.animation_.cancel(self);
        self.animation_ = Animation(_elevation = self.elevation,
            d=0.08);
        self.animation_.start(self)

MDFloatLayout:

    WidgetWithShadow:
        size: [root.size[1] / 2] * 2
        pos_hint: {"center": [0.5, 0.5]}
'''


class WidgetWithShadow(
    CircularElevationBehavior,
    CircularRippleBehavior,
    ButtonBehavior,
    MDBBoxLayout,
```

(continues on next page)

(continued from previous page)

```

):

def __init__(self, **kwargs):
    # always set the elevation before the super().__init__ call
    # self.elevation = 6
    super().__init__(**kwargs)

def on_size(self, *args):
    self.radius = [self.size[0] / 2]

class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)

Example().run()

```

angle

Angle of rotation in degrees of the current shadow. This value is shared across different widgets.

Note: This value will affect both, hard and soft shadows. Each shadow has his own origin point that's computed every time the elevation changes.

Warning: Do not add *PushMatrix* inside the canvas before and add *PopMatrix* in the next layer, this will cause visual errors, because the stack used will clip the push and pop matrix already inside the `canvas.before` layer.

Incorrect:

```

<TiltedWidget>
    canvas.before:
        PushMatrix
        [...]
    canvas:
        PopMatrix

```

Correct:

```

<TiltedWidget>
    canvas.before:
        PushMatrix
        [...]
    canvas:
        PopMatrix

```

angle is an `NumericProperty` and defaults to 0.

radius

Radius of the corners of the shadow. This values represents each corner of the shadow, starting from *top-left* corner and going clockwise.

```
radius = [
    "top-left",
    "top-right",
    "bottom-right",
    "bottom-left",
]
```

This value can be expanded thus allowing this settings to be valid:

```
widget.radius=[0] # Translates to [0, 0, 0, 0]
widget.radius=[10, 3] # Translates to [10, 3, 10, 3]
widget.radius=[7.0, 8.7, 1.5, 3.0] # Translates to [7, 8, 1, 3]
```

Note: This value will affect both, hard and soft shadows. This value only affects *RoundedRectangularElevationBehavior* for now, but can be stored and used by custom shadow draw functions.

radius is an [VariableListProperty](#) and defaults to *[0, 0, 0, 0]*.

shadow_pos

Custom shadow origin point. If this property is set, *_shadow_pos* will be omitted.

This property allows users to fake light source.

shadow_pos is an [ListProperty](#) and defaults to *[0, 0]*.

Note: this value overwrite the *_shadow_pos* processing.

shadow_group

Widget's shadow group. By default every widget with a shadow is saved inside the memory *__shadow_groups* as a weakref. This means that you can have multiple light sources, one for every shadow group.

To fake a light source use *force_shadow_pos*.

shadow_group is an [StringProperty](#) and defaults to “*global*”.

soft_shadow_size

Size of the soft shadow texture over the canvas.

soft_shadow_size is an [ListProperty](#) and defaults to *[0, 0]*.

Note: This property is automatically processed.

soft_shadow_pos

Position of the hard shadow texture over the canvas.

soft_shadow_pos is an [ListProperty](#) and defaults to *[0, 0]*.

Note: This property is automatically processed.

soft_shadow_cl

Color of the soft shadow.

`soft_shadow_cl` is an `ListProperty` and defaults to `[0, 0, 0, 0.15]`.

hard_shadow_texture

Texture of the hard shadow texture for the canvas.

`hard_shadow_texture` is an `Image` and defaults to `None`.

Note: This property is automatically processed when elevation is changed.

hard_shadow_size

Size of the hard shadow texture over the canvas.

`hard_shadow_size` is an `ListProperty` and defaults to `[0, 0]`.

Note: This property is automatically processed when elevation is changed.

hard_shadow_pos

Position of the hard shadow texture over the canvas.

`hard_shadow_pos` is an `ListProperty` and defaults to `[0, 0]`.

Note: This property is automatically processed when elevation is changed.

hard_shadow_cl

Color of the hard shadow.

Note: `hard_shadow_cl` is an `ListProperty` and defaults to `[0, 0, 0, 0.15]`.

hard_shadow_offset

This value sets a special offset to the shadow canvas, this offset allows a correct draw of the canvas size. allowing the effect to correctly blur the image in the given space.

`hard_shadow_offset` is an `BoundedNumericProperty` and defaults to 2.

soft_shadow_offset

This value sets a special offset to the shadow canvas, this offset allows a correct draw of the canvas size. allowing the effect to correctly blur the image in the given space.

`soft_shadow_offset` is an `BoundedNumericProperty` and defaults to 4.

draw_shadow

This property controls the draw call of the context.

This property is automatically set to `__draw_shadow__` inside the `super().__init__` call. unless the property is different of None.

To set a different drawing instruction function, set this property before the `super().__init__` call inside the `__init__` definition of the new class.

You can use the source for this classes as example of how to draw over with the context:

Real time shadows:

1. `RectangularElevationBehavior`
2. `CircularElevationBehavior`

3. *RoundedRectangularElevationBehavior*
4. *ObservableShadow*

Fake shadows (don't use this property):

1. *FakeRectangularElevationBehavior*
2. *FakeCircularElevationBehavior*

`draw_shadow` is an `ObjectProperty` and defaults to `None`.

Note: If this property is left to `None` the `CommonElevationBehavior` will set to a function that will raise a `NotImplementedError` inside `super().__init__`.

Follow the next example to set a new draw instruction for the class inside `__init__`:

```
class RoundedRectangularElevationBehavior(CommonElevationBehavior):  
    """  
        Shadow class for the RoundedRectangular shadow behavior.  
        Controls the size and position of the shadow.  
    """  
  
    def __init__(self, **kwargs):  
        self._draw_shadow = WeakMethod(self.__draw_shadow__)  
        super().__init__(**kwargs)  
  
    def __draw_shadow__(self, origin, end, context=None):  
        context.draw(...)
```

Context is a `Pillow ImageDraw` class. For more information check the [Pillow official documentation](<https://github.com/python-pillow/Pillow/>).

on_shadow_group(*self, instance, value*)

This function controls the shadow group of the widget. Do not use Directly to change the group. instead, use the `shadow_group` property.

force_shadow_pos(*self, shadow_pos*)

This property forces the shadow position in every widget inside the widget. The argument `shadow_pos` is expected as a <class ‘list’> or <class ‘tuple’>.

update_group_property(*self, property_name, value*)

This functions allows to change properties of every widget inside the shadow group.

shadow_preset(*self, *args*)

This function is meant to set the default configuration of the elevation.

After a new instance is created, the elevation property will be launched and thus this function will update the elevation if the KV lang have not done it already.

Works similar to an `__after_init__` call inside a widget.

on_elevation(*self, instance, value*)

Elevation event that sets the current elevation value to `_elevation`.

on_disabled(*self, instance, value*)

This function hides the shadow when the widget is disabled. It sets the shadow to `0`.

on__shadow_pos(*self, ins, val*)

Updates the shadow with the computed value.

Call this function every time you need to force a shadow update.

`on_shadow_pos(self, ins, val)`

Updates the shadow with the fixed value.

Call this function every time you need to force a shadow update.

`class kivymd.uix.behaviors.elevation.RectangularElevationBehavior(**kwargs)`

Base class for a rectangular elevation behavior.

`class kivymd.uix.behaviors.elevation.CircularElevationBehavior(**kwargs)`

Base class for a circular elevation behavior.

`class kivymd.uix.behaviors.elevation.RoundedRectangularElevationBehavior(**kwargs)`

Base class for rounded rectangular elevation behavior.

`class kivymd.uix.behaviors.elevation.ObservableShadow(**kwargs)`

`ObservableShadow` is real time shadow render that it's intended to only render a partial shadow of widgets based upon on the window observable area, this is meant to improve the performance of bigger widgets.

Warning: This is an empty class, the name has been reserved for future use. if you include this clas in your object, you wil get a *NotImplementedError*.

`class kivymd.uix.behaviors.elevation.FakeRectangularElevationBehavior(**kwargs)`

FakeRectangularElevationBehavior is a shadow mockup for widgets. Improves performance using cached images inside `kivymd.images` dir

This class cast a fake Rectangular shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(
    ThemableBehavior,
    FakeCircularElevationBehavior,
    SpecificBackgroundColorBehavior,
    # here you add the other front end classes for the widget front_end,
):  
    [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

FakeCircularElevationBehavior will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class after the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_rectangular_Card(
    MDCard,
    FakeRectangularElevationBehavior
):  
    [...]
```

Note: About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners.

```
class kivymd.uix.behaviors.elevation.FakeCircularElevationBehavior(**kwargs)
```

FakeCircularElevationBehavior is a shadow mockup for widgets. Improves performance using cached images inside *kivymd.images* dir

This class cast a fake elliptic shadow behaind the widget.

You can either use this behavior to overwrite the elevation of a prefab widget, or use it directly inside a new widget class definition.

Use this class as follows for new widgets:

```
class NewWidget(  
    ThemableBehavior,  
    FakeCircularElevationBehavior,  
    SpecificBackgroundColorBehavior,  
    # here you add the other front end classes for the widget front_end,  
):  
    [...]
```

With this method each class can draw it's content in the canvas in the correct order, avoiding some visual errors.

FakeCircularElevationBehavior will load prefabricated textures to optimize loading times.

Also, this class allows you to overwrite real time shadows, in the sence that if you are using a standard widget, like a button, MDCard or Toolbar, you can include this class afer the base class to optimize the loading times.

As an example of this flexibility:

```
class Custom_Circular_Card(  
    MDCard,  
    FakeCircularElevationBehavior  
):  
    [...]
```

Note: About rounded corners: be careful, since this behavior is a mockup and will not draw any rounded corners. only perfect ellipses.

2.4.3 Focus

Changing the background color when the mouse is on the widget.

To apply focus behavior, you must create a new class that is inherited from the widget to which you apply the behavior and from the `FocusBehavior` class.

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import RectangularElevationBehavior, FocusBehavior
from kivymd.uix.boxlayout import MDBBoxLayout

KV = """
MDScreen:
    md_bg_color: 1, 1, 1, 1

    FocusWidget:
        size_hint: .5, .3
        pos_hint: {"center_x": .5, "center_y": .5}
        md_bg_color: app.theme_cls.bg_light

        MDLabel:
            text: "Label"
            theme_text_color: "Primary"
            pos_hint: {"center_y": .5}
            halign: "center"
    ...
"""

class FocusWidget(MDBBoxLayout, RectangularElevationBehavior, FocusBehavior):
    pass

class Test(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

Test().run()
```

Color change at focus/defocus

```
FocusWidget:
    focus_color: 1, 0, 1, 1
    unfocus_color: 0, 0, 1, 1
```

API - kivymd.uix.behaviors.focus_behavior

```
class kivymd.uix.behaviors.FocusBehavior(**kwargs)
```

Events

on_enter Called when mouse enters the bbox of the widget AND the widget is visible

on_leave Called when the mouse exits the widget AND the widget is visible

focus_behavior

Using focus when hovering over a widget.

focus_behavior is a `BooleanProperty` and defaults to *False*.

focus_color

The color of the widget when the mouse enters the bbox of the widget.

focus_color is a `ColorProperty` and defaults to *None*.

unfocus_color

The color of the widget when the mouse exits the bbox of the widget.

unfocus_color is a `ColorProperty` and defaults to *None*.

on_enter(self)

Called when mouse enter the bbox of the widget.

on_leave(self)

Called when the mouse exit the widget.

2.4.4 Ripple

Classes implements a circular and rectangular ripple effects.

To create a widget with ircular ripple effect, you must create a new class that inherits from the `CircularRippleBehavior` class.

For example, let's create an image button with a circular ripple effect:

```
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior
from kivy.uix.image import Image

from kivymd.app import MDApp
from kivymd.uix.behaviors import CircularRippleBehavior

KV = '''
#:import images_path kivymd.images_path

Screen:

    CircularRippleButton:
```

(continues on next page)

(continued from previous page)

```

source: f"images_path}/kivymd.png"
size_hint: None, None
size: "250dp", "250dp"
pos_hint: {"center_x": .5, "center_y": .5}
...

class CircularRippleButton(CircularRippleBehavior, ButtonBehavior, Image):
    def __init__(self, **kwargs):
        self.ripple_scale = 0.85
        super().__init__(**kwargs)

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

Example().run()

```

To create a widget with rectangular ripple effect, you must create a new class that inherits from the `RectangularRippleBehavior` class:

```

from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import RectangularRippleBehavior, BackgroundColorBehavior

KV = '''
Screen:

    RectangularRippleButton:
        size_hint: None, None
        size: "250dp", "50dp"
        pos_hint: {"center_x": .5, "center_y": .5}
'''

class RectangularRippleButton(
    RectangularRippleBehavior, ButtonBehavior, BackgroundColorBehavior
):
    md_bg_color = [0, 0, 1, 1]

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

```

(continues on next page)

(continued from previous page)

```
Example().run()
```

API - kivymd.uix.behaviors.ripple_behavior

class kivymd.uix.behaviors.ripple_behavior.CommonRipple

Base class for ripple effect.

ripple_rad_default

Default value of the ripple effect radius.

ripple_rad_default is an `NumericProperty` and defaults to *1*.

ripple_color

Ripple color in `rgba` format.

ripple_color is an `ColorProperty` and defaults to *None*.

ripple_alpha

Alpha channel values for ripple effect.

ripple_alpha is an `NumericProperty` and defaults to *0.5*.

ripple_scale

Ripple effect scale.

ripple_scale is an `NumericProperty` and defaults to *None*.

ripple_duration_in_fast

Ripple duration when touching to widget.

ripple_duration_in_fast is an `NumericProperty` and defaults to *0.3*.

ripple_duration_in_slow

Ripple duration when long touching to widget.

ripple_duration_in_slow is an `NumericProperty` and defaults to *2*.

ripple_duration_out

The duration of the disappearance of the wave effect.

ripple_duration_out is an `NumericProperty` and defaults to *0.3*.

ripple_func_in

Type of animation for ripple in effect.

ripple_func_in is an `StringProperty` and defaults to *'out_quad'*.

ripple_func_out

Type of animation for ripple out effect.

ripple_func_out is an `StringProperty` and defaults to *'ripple_func_out'*.

abstract lay_canvas_instructions(self)

```

start_ripple(self)
finish_ripple(self)
fade_out(self, *args)
anim_complete(self, *args)
on_touch_down(self, touch)
on_touch_move(self, touch, *args)
on_touch_up(self, touch)

class kivymd.uix.behaviors.ripple_behavior.RectangularRippleBehavior
    Class implements a rectangular ripple effect.

    ripple_scale
        See ripple\_scale.
        ripple_scale is an NumericProperty and defaults to 2.75.

    lay_canvas_instructions(self)

class kivymd.uix.behaviors.ripple_behavior.CircularRippleBehavior
    Class implements a circular ripple effect.

    ripple_scale
        See ripple\_scale.
        ripple_scale is an NumericProperty and defaults to 1.

    lay_canvas_instructions(self)

```

2.4.5 ToggleButton

This behavior must always be inherited after the button's Widget class since it works with the inherited properties of the button class.

example:

```
class MyToggleButtonWidget(MDFlatButton, MDToggleButton):
    # [...]
    pass
```

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors.toggle_behavior import MDToggleButton
from kivymd.uix.button import MDRectangleFlatButton

KV = '''
Screen:

    MDBoxLayout:
        adaptive_size: True
        pos_hint: {"center_x": .5, "center_y": .5}

```

(continues on next page)

(continued from previous page)

```

MyToggleButton:
    text: "Show ads"
    group: "x"

MyToggleButton:
    text: "Do not show ads"
    group: "x"

MyToggleButton:
    text: "Does not matter"
    group: "x"
    ...

class MyToggleButton(MDRectangleFlatButton, MDToggleButton):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.background_down = self.theme_cls.primary_light

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()

```

```

class MyToggleButton(MDFillRoundFlatButton, MDToggleButton):
    def __init__(self, **kwargs):
        self.background_down = MDApp.get_running_app().theme_cls.primary_dark
        super().__init__(**kwargs)

```

You can inherit the **MyToggleButton** class only from the following classes

- *MDRaisedButton*
- *MDFlatButton*
- *MDRectangleFlatButton*
- *MDRectangleFlatIconButton*
- *MDRoundFlatButton*
- *MDRoundFlatIconButton*
- *MDFillRoundFlatButton*
- *MDFillRoundFlatIconButton*

API - kivymd.uix.behaviors.toggle_behavior

```
class kivymd.uix.behaviors.toggle_behavior.MDToggleButton(**kwargs)
```

This [mixin](#) class provides [togglebutton](#) behavior. Please see the [togglebutton behaviors](#) module documentation for more information.

New in version 1.8.0.

background_normal

Color of the button in `rgba` format for the ‘normal’ state.

background_normal is a [ColorProperty](#) and is defaults to *None*.

background_down

Color of the button in `rgba` format for the ‘down’ state.

background_down is a [ColorProperty](#) and is defaults to *None*.

font_color_normal

Color of the font’s button in `rgba` format for the ‘normal’ state.

font_color_normal is a [ColorProperty](#) and is defaults to *None*.

font_color_down

Color of the font’s button in `rgba` format for the ‘down’ state.

font_color_down is a [ColorProperty](#) and is defaults to *[1, 1, 1, 1]*.

2.4.6 Magic

Magical effects for buttons.

Warning: Magic effects do not work correctly with *KivyMD* buttons!

To apply magic effects, you must create a new class that is inherited from the widget to which you apply the effect and from the [MagicBehavior](#) class.

In *KV file*:

```
<MagicButton@MagicBehavior+MDRectangleFlatButton>
```

In *python file*:

```
class MagicButton(MagicBehavior, MDRectangleFlatButton):
    pass
```

The `MagicBehavior` class provides five effects:

- `MagicBehavior.wobble`
- `MagicBehavior.grow`
- `MagicBehavior.shake`
- `MagicBehavior.twist`
- `MagicBehavior.shrink`

Example:

```
from kivymd.app import MDApp
from kivy.lang import Builder

KV = '''
#:import MagicBehavior kivymd.uix.behaviors.MagicBehavior

<MagicButton@MagicBehavior+MDRectangleFlatButton>

FloatLayout:

    MagicButton:
        text: "WOBBLE EFFECT"
        on_release: self.wobble()
        pos_hint: {"center_x": .5, "center_y": .3}

    MagicButton:
        text: "GROW EFFECT"
        on_release: self.grow()
        pos_hint: {"center_x": .5, "center_y": .4}

    MagicButton:
        text: "SHAKE EFFECT"
        on_release: self.shake()
        pos_hint: {"center_x": .5, "center_y": .5}

    MagicButton:
        text: "TWIST EFFECT"
        on_release: self.twist()
        pos_hint: {"center_x": .5, "center_y": .6}

    MagicButton:
        text: "SHRINK EFFECT"
        on_release: self.shrink()
        pos_hint: {"center_x": .5, "center_y": .7}
'''


class Example(MDApp):
    def build(self):
        return Builder.load_string(KV)
```

(continues on next page)

(continued from previous page)

```
Example().run()
```

API - kivymd.uix.behaviors.magic_behavior

```
class kivymd.uix.behaviors.magic_behavior.MagicBehavior

magic_speed
    Animation playback speed.
    magic_speed is a NumericProperty and defaults to 1.

grow(self)
    Grow effect animation.

shake(self)
    Shake effect animation.

wobble(self)
    Wobble effect animation.

twist(self)
    Twist effect animation.

shrink(self)
    Shrink effect animation.

on_touch_up(self, *args)
```

2.4.7 Touch

Provides easy access to events.

The following events are available:

- on_long_touch
- on_double_tap
- on_triple_tap

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import TouchBehavior
from kivymd.uix.button import MDRaisedButton

KV = '''
Screen:

    MyButton:
        text: "PRESS ME"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class MyButton(MDRaisedButton, TouchBehavior):
    def on_long_touch(self, *args):
        print("<on_long_touch> event")

    def on_double_tap(self, *args):
        print("<on_double_tap> event")

    def on_triple_tap(self, *args):
        print("<on_triple_tap> event")


class MainApp(MDApp):
    def build(self):
        return Builder.load_string(KV)

MainApp().run()
```

API - `kivymd.uix.behaviors.touch_behavior`

```
class kivymd.uix.behaviors.touch_behavior.TouchBehavior(**kwargs)

duration_long_touch
    Time for a long touch.

    duration_long_touch is an NumericProperty and defaults to 0.4.

create_clock(self, widget, touch, *args)
delete_clock(self, widget, touch, *args)
on_long_touch(self, touch, *args)
    Called when the widget is pressed for a long time.

on_double_tap(self, touch, *args)
    Called by double clicking on the widget.
```

on_triple_tap(self, touch, *args)
Called by triple clicking on the widget.

2.4.8 Background Color

Note: The following classes are intended for in-house use of the library.

API - kivymd.uix.behaviors.backgroundcolor_behavior

class kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior(kwargs)**
Common base class for rectangular and circular elevation behavior.

background

Background image path.

background is a **StringProperty** and defaults to *None*.

r

The value of red in the rgba palette.

r is an **BoundedNumericProperty** and defaults to *1.0*.

g

The value of green in the rgba palette.

g is an **BoundedNumericProperty** and defaults to *1.0*.

b

The value of blue in the rgba palette.

b is an **BoundedNumericProperty** and defaults to *1.0*.

a

The value of alpha channel in the rgba palette.

a is an **BoundedNumericProperty** and defaults to *0.0*.

radius

Canvas radius.

```
# Top left corner slice.
MDBoxLayout:
    md_bg_color: app.theme_cls.primary_color
    radius: [25, 0, 0, 0]
```

radius is an **VariableListProperty** and defaults to *[0, 0, 0, 0]*.

md_bg_color

The background color of the widget (**Widget**) that will be inherited from the *BackgroundColorBehavior* class.

For example:

```
Widget:
    canvas:
        Color:
            rgba: 0, 1, 1, 1
    Rectangle:
        size: self.size
        pos: self.pos
```

similar to code:

```
<MyWidget@BackgroundColorBehavior>
    md_bg_color: 0, 1, 1, 1
```

md_bg_color is an [ReferenceListProperty](#) and defaults to *r, g, b, a*.

line_color

If a custom value is specified for the *line_color parameter*, the border of the specified color will be used to border the widget:

```
MDBoxLayout:
    size_hint: .5, .2
    md_bg_color: 0, 1, 1, .5
    line_color: 0, 0, 1, 1
    radius: [24, ]
```

New in version 0.104.2.

line_color is an [ColorProperty](#) and defaults to *[0, 0, 0, 0]*.

angle

background_origin

`update_background_origin(self, *args)`

```
class kivymd.uix.behaviors.backgroundcolor_behavior.SpecificBackgroundColorBehavior(**kwargs)
Common base class for rectangular and circular elevation behavior.
```

background_palette

See [kivymd.color_definitions.palette](#).

background_palette is an [OptionProperty](#) and defaults to ‘Primary’.

background_hue

See [kivymd.color_definitions.hue](#).

background_hue is an [OptionProperty](#) and defaults to ‘500’.

specific_text_color

specific_text_color is an [ColorProperty](#) and defaults to *[0, 0, 0, 0.87]*.

specific_secondary_text_color

specific_secondary_text_color is an :class:`~kivy.properties.ColorProperty` and defaults to *[0, 0, 0, 0.87]*.

2.5 Changelog

2.5.1 Unreleased

See on GitHub: [branch master](#) | [compare 0.104.2/master](#)

```
pip install https://github.com/kivymd/KivyMD/archive/master.zip
```

- Bug fixes and other minor improvements.
- Added `ImageLeftWidgetWithoutTouch`, `ImageRightWidgetWithoutTouch`, `IconRightWidgetWithoutTouch`, `IconLeftWidgetWithoutTouch` classes to `kivymd/uix/list.py` module;
- Added `MDStepper` component;
- Added a feature, `show_disks` to the `MDFFileManager` class, that allows you to display the disks and folders contained in them;

2.5.2 0.104.2

See on GitHub: [tag 0.104.2](#) | [compare 0.104.1/0.104.2](#)

```
pip install kivymd==0.104.2
```

- Bug fixes and other minor improvements.
- Add `HotReloadViewer` class
- Added features to `Snackbar` class: use padding, set custom button color, elevation
- Add `MDToggleButton` class
- Change to *Material Design Baseline* dark theme spec
- Fix *ReferenceError: weakly-referenced object no longer exists* when start demo application
- Changed the default value for the `theme_text_color` parameter in the `BaseButton` class (to the value “*Primary*”)
- Fix setting of the `text_color_normal` and `text_color_active` parameters - earlier their values did not affect anything
- Fixed the length of the right edge of the border in relation to the hint text when the `MDTextField` is in the `rectangle` mode
- Add `get_tab_list` method to `MDTabs` class
- Add hover behavior when using `MDDropdownMenu` class
- Added the feature to use the `FitImage` component to download images from the network
- The `elevation` value for `RectangularElevationBehavior` and `CircularElevationBehavior` classes after pressing was always set to 2 - fixed
- Methods that implement the ripple effect have always been called twice - fixed
- The `SmartTile` class now uses the `FitImage` class to display images instead of the `Image` class
- Removed dependency on `PIL` library
- Add `hint_bg_color`, `hint_text_color`, `hint_radius` attributes to `MDSlider` class
- Delete `progressloader.py`
- Delete `context_menu.py`

- Added the feature to control the properties of menu items during creation in *MDDropdownMenu* class
- Added the feature to change the number of buttons after creating the *MDFloatingActionButtonSpeedDial* object
- Added the feature to set the *font_name* property for the *MDTabsLabel* class
- Add *MDCarousel* class
- Delete *kivymd/uix/useranimationcard.py*
- Added usage types for *MNavigationDrawer* class: *modal/standard*
- Added stencil instructions to the *FitImage* class canvas
- Added *on_ref_press* and *switch_tab* methods to *MDTabs* class
- Added *on_release* method for menu item events instead of callback method to *MDDropdownMenu* class
- Added *palette* attribute - the feature to change the color of the *MDSpinner* when changing rotation cycles
- Added the feature to change the border color of the *MDRectangleFlatIconButton* class
- Add *MDRelativeLayout* class
- Added the feature to use radius for *MNavigationDrawer* corners
- Removed *UserAnimationCard* class
- Added feature to set background color for *MDialog* class
- Added *MNavigationRail* component
- Added *MDSwiper* component
- Added ripple effect to *MDTabs* class
- Added the feature to set toast positions on an *Android* device
- Added of tooltips to *MDToolbar* icons
- Fixed *MDBottomAppBar* notch transparency
- Updated *MDatePicker* class to material design specification.
- Updated *MDTimePicker* class to material design specification.
- Elevation behavior redesign to comply with the material design specification.
- Removed the *vendor* package.
- Added the feature to use a class instance (*Kivy* or *KivyMD* widget), which will be added to the *MDDropdownMenu* class menu header.

2.5.3 0.104.1

See on GitHub: [tag 0.104.1](#) | [compare 0.104.0/0.104.1](#)

```
pip install kivymd==0.104.1
```

- Bug fixes and other minor improvements.
- Added *MDGridLayout* and *MDBBoxLayout* classes
- Add *TouchBehavior* class
- Add *radius* parameter to *BackgroundColorBehavior* class
- Add *MDScreen* class

- Add *MDFloatLayout* class
- Added a *MTextField* with *fill* mode
- Added a shadow, increased speed of opening, added the feature to control the position of the *MDDropdownMenu* class
- The *MDDropDownItem* class is now a regular element, such as a button
- Added the ability to use the texture of the icon on the right in any *MTextField* classes
- Added the feature to use ripple and focus behavior in *MDCard* class
- *MDDialogs* class redesigned to meet material design requirements
- Added *MDataTable* class

2.5.4 0.104.0

See on GitHub: [tag 0.104.0](#) | [compare 0.103.0/0.104.0](#)

```
pip install kivymd==0.104.0
```

- Fixed bug in *kivymd.uix.expansionpanel.MDEExpansionPanel* if, with the panel open, without closing it, try to open another panel, then the chevron of the first panel remained open.
- The *kivymd.uix.textfield.MDTextFieldRound* class is now directly inherited from the *kivy.uix.textinput.TextInput* class.
- Removed *kivymd.uix.textfield.MDTextFieldClear* class.
- *kivymd.uix.navigationdrawer.NavigationLayout* allowed to add *kivymd.uix.toolbar.MToolbar* class.
- Added feature to control range of dates to be active in *kivymd.uix.picker.MDDatePicker* class.
- Updated *kivymd.uix.navigationdrawer.MDNavigationDrawer* realization.
- Removed *kivymd.uix.card.MDCardPost* class.
- Added *kivymd.uix.card.MDCardSwipe* class.
- Added *switch_tab* method for switching tabs to *kivymd.uix.bottomnavigationMDBottomNavigation* class.
- Added feature to use panel type in the *kivymd.uix.expansionpanel.MDEExpansionPanel* class: *kivymd.uix.expansionpanel.MDEExpansionPanelOneLine*, *kivymd.uix.expansionpanel.MDEExpansionPanelTwoLine* or *kivymd.uix.expansionpanel.MDEExpansionPanelThreeLine*.
- Fixed panel opening animation in the *kivymd.uix.expansionpanel.MDEExpansionPanel* class.
- Delete *kivymd.uix.managerswiper.py*
- Add *MDFloatingActionButtonSpeedDial* class
- Added the feature to create text on tabs using markup, thereby triggering the *on_ref_press* event in the *MDTabLabel* class
- Added *color_indicator* attribute to set custom indicator color in the *MDTabs* class
- Added the feature to change the background color of menu items in the *BaseListItem* class
- Add *MDTapTargetView* class

2.5.5 0.103.0

See on GitHub: [tag 0.103.0](#) | [compare 0.102.1/0.103.0](#)

```
pip install kivymd==0.103.0
```

- Fix *MDSwitch* size according to *material design* guides
- Fix *MDSwitch*'s thumb position when size changes
- Fix position of the icon relative to the right edge of the *MDChip* class on mobile devices
- Updated *MDBottomAppBar* class.
- Updated *navigationdrawer.py*
- Added *on_tab_switch* method that is called when switching tabs (*MDTabs* class)
- Added *FpsMonitor* class
- Added *fitimage.py* - feature to automatically crop a *Kivy* image to fit your layout
- Added animation when changing the action button position mode in *MDBottomAppBar* class
- Delete *fanscreenmanager.py*
- Bug fixes and other minor improvements.

2.5.6 0.102.1

See on GitHub: [tag 0.102.1](#) | [compare 0.102.0/0.102.1](#)

```
pip install kivymd==0.102.1
```

- Implemented the ability [Backdrop](<https://material.io/components/backdrop>)
- Added *MDApp* class. Now app object should be inherited from *kivymd.app.MDApp*.
- Added *MDRoundImageButton* class.
- Added *MDTooltip* class.
- Added *MDBanner* class.
- Added hook for *PyInstaller* (add *hookspath=[kivymd.hooks_path]*).
- Added examples of *spec* files for building [Kitchen Sink demo](https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink).
- Added some features to *MDProgressLoader*.
- Added feature to preview the current value of *MDSlider*.
- Added feature to use custom screens for dialog in *MDBottomSheet* class.
- Removed *MDPopupScreen*.
- Added [studies](https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink/studies) directory for demos in Material Design.
- Bug fixes and other minor improvements.

2.5.7 0.102.0

See on GitHub: [tag 0.102.0](#) | compare 0.101.8/0.102.0

```
pip install kivymd==0.102.0
```

- Moved *kivymd.behaviors* to *kivymd.uix.behaviors*.
- Updated [Iconic font] (<https://github.com/Templarian/MaterialDesign-Webfont>) (v4.5.95).
- Added *blank* icon to *icon_definitions*.
- Bug fixes and other minor improvements.

2.5.8 0.101.8

See on GitHub: [tag 0.101.8](#) | compare 0.101.7/0.101.8

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.8.zip
```

- Added *uix* and *behaviors* folder to *package_data*.

2.5.9 0.101.7

See on GitHub: [tag 0.101.7](#) | compare 0.101.6/0.101.7

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.7.zip
```

- Fixed colors and position of the buttons in the *Buttons* demo screen ([Kitchen Sink demo] (https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink)).
- Displaying percent of loading kv-files ([Kitchen Sink demo] (https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink))).

2.5.10 0.101.6

See on GitHub: [tag 0.101.6](#) | compare 0.101.5/0.101.6

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.6.zip
```

- Fixed *NameError: name 'MDThemePicker' is not defined*.

2.5.11 0.101.5

See on GitHub: [tag 0.101.5](#) | compare 0.101.4/0.101.5

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.5.zip
```

- Added feature to see source code of current example ([Kitchen Sink demo] (https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink))).
- Added names of authors of this fork ([Kitchen Sink demo] (https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink))).
- Bug fixes and other minor improvements.

2.5.12 0.101.4

See on GitHub: [tag 0.101.4](#) | [compare 0.101.3/0.101.4](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.4.zip
```

- Bug fixes and other minor improvements.

2.5.13 0.101.3

See on GitHub: [tag 0.101.3](#) | [compare 0.101.2/0.101.3](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.3.zip
```

- Bug fixes and other minor improvements.

2.5.14 0.101.2

See on GitHub: [tag 0.101.2](#) | [compare 0.101.1/0.101.2](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.2.zip
```

- Bug fixes and other minor improvements.

2.5.15 0.101.1

See on GitHub: [tag 0.101.1](#) | [compare 0.101.0/0.101.1](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.1.zip
```

- Bug fixes and other minor improvements.

2.5.16 0.101.0

See on GitHub: [tag 0.101.0](#) | [compare 0.100.2/0.101.0](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.0.zip
```

- Added *MDContextMenu* class.
- Added *MDExpansionPanel* class.
- Removed *MDAccordion* and *MDAccordionListItem*. Use *MDExpansionPanel* instead.
- Added *HoverBehavior* class by [Olivier POYEN](<https://gist.github.com/opqopq/15c707dc4cffc2b6455f>).
- Added markup support for buttons.
- Added *duration* property to *Toast*.
- Added *TextInput*'s events and properties to *MDTextFieldRound*.
- Added feature to resize text field
- Added color property to *MDSeparator* class

- Added [tool](https://github.com/kivymd/KivyMD/blob/master/kivymd/tools/update_icons.py) for updating [Iconic font](<https://github.com/Templarian/MaterialDesign-Webfont>).
- Updated [Iconic font](<https://github.com/Templarian/MaterialDesign-Webfont>) (v4.3.95).
- Added new examples for [Kitchen Sink demo](https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink).
- Bug fixes and other minor improvements.

2.5.17 0.100.2

See on GitHub: [tag 0.100.2](#) | [compare 0.100.1/0.100.2](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.100.2.zip
```

- [Black](<https://github.com/psf/black>) formatting.

2.5.18 0.100.1

See on GitHub: [tag 0.100.1](#) | [compare 0.100.0/0.100.1](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.100.1.zip
```

- *MDUserAnimationCard* uses *Image* instead of *AsyncImage*.

2.5.19 0.100.0

See on GitHub: [tag 0.100.0](#) | [compare 0.99.99/0.100.0](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.100.0.zip
```

- Added feature to change color for *MDStackFloatingButtons*.

2.5.20 0.99.99.01

See on GitHub: [tag 0.99.99.01](#) | [compare 0.99.98/0.99.99.01](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.99.01.zip
```

- Fixed *MDDrawer*.*use_logo*.

2.5.21 0.99.99

See on GitHub: [tag 0.99.99](#) | [compare 0.99.99.01/0.99.99](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.99.zip
```

- Added *icon_color* property for *NavigationDrawerIconButton*.

2.5.22 0.99.98

See on GitHub: [tag 0.99.98](#) | [compare 0.99.97/0.99.98](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.98.zip
```

- Added *MDFillRoundFlatButton* class.

2.5.23 0.99.97

See on GitHub: [tag 0.99.97](#) | [compare 0.99.96/0.99.97](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.97.zip
```

- Fixed *Spinner* animation.

2.5.24 0.99.96

See on GitHub: [tag 0.99.96](#) | [compare 0.99.95/0.99.96](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.96.zip
```

- Added *asynckivy* module by [Nattōsai Mitō](<https://github.com/gottadiveintopython/asynckivy>).

2.5.25 0.99.95

See on GitHub: [tag 0.99.95](#) | [compare 0.99.94/0.99.95](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.95.zip
```

- Added function to create a round image in *kivymd/utils/cropimage.py* module.
- Added *MDCustomRoundIconButton* class.
- Added demo application [Account Page](<https://www.youtube.com/watch?v=dfUOwqtYoYg>) for [Kitchen Sink demo](https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink).

2.5.26 0.99.94

See on GitHub: [tag 0.99.94](#) | [compare 0.99.93/0.99.94](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.94.zip
```

- Added *_no_ripple_effect* property to *BaseListItem* class.
- Added check to use *ripple effect* in *RectangularRippleBehavior* class.
- [Disabled](https://www.youtube.com/watch?v=P_9oSx0Pz_U) using *ripple effect* in *MADAccordionListItem* class.

2.5.27 0.99.93

See on GitHub: [tag 0.99.93](#) | [compare 0.99.92/0.99.93](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.93.zip
```

- Updated [Iconic font](<https://github.com/Templarian/MaterialDesign-Webfont>) (v3.6.95).

2.5.28 0.99.92

See on GitHub: [tag 0.99.92](#) | [compare 0.99.91/0.99.92](#)

```
pip install https://github.com/kivymd/KivyMD/archive/0.99.92.zip
```

- Removed automatic change of text field length in *MDFTextFieldRound* class.

2.6 About

2.6.1 License

Refer to [LICENSE](#).

MIT License

Copyright (c) 2015 Andrés Rodríguez and other contributors - KivyMD library up to ↵ version 0.1.2

Copyright (c) 2021 KivyMD Team and other contributors - KivyMD library version 0.1.3 and ↵ higher

Other libraries used in the project:

Copyright (c) 2010-2021 Kivy Team and other contributors
 Copyright (c) 2013 Brian Knapp - Androidoast library
 Copyright (c) 2014 LogicalDash - stiffscroll library
 Copyright (c) 2015 Kivy Garden - tabs module
 Copyright (c) 2020 Nattōsai Mitō - asynckivy module
 Copyright (c) 2021 tshirtman - magic_behavior module
 Copyright (c) 2021 shashi278 - taptargetview module
 Copyright (c) 2020 Benedikt Zwölfer - fitimage module

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

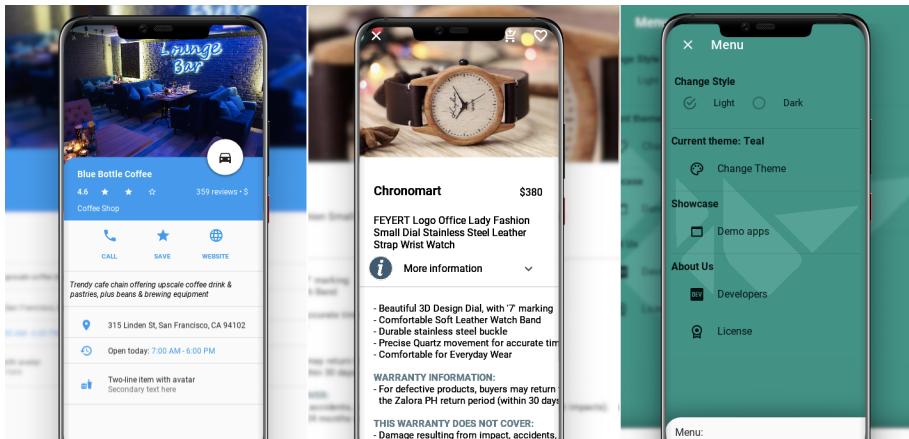
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

(continues on next page)

(continued from previous page)

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.7 KivyMD



Is a collection of Material Design compliant widgets for use with, Kivy cross-platform graphical framework a framework for cross-platform, touch-enabled graphical applications. The project's goal is to approximate Google's [Material Design spec](#) as close as possible without sacrificing ease of use or application performance.

This library is a fork of the [KivyMD](#) project the author of which stopped supporting this project three years ago. We found the strength and brought this project to a new level. Currently we're in **beta** status, so things are changing all the time and we cannot promise any kind of API stability. However it is safe to vendor now and make use of what's currently available.

Join the project! Just fork the project, branch out and submit a pull request when your patch is ready. If any changes are necessary, we'll guide you through the steps that need to be done via PR comments or access to your for may be requested to outright submit them. If you wish to become a project developer (permission to create branches on the project without forking for easier collaboration), have at least one PR approved and ask for it. If you contribute regularly to the project the role may be offered to you without asking too.

2.7.1 API - kivymd

kivymd.release = False

kivymd.path

Path to KivyMD package directory.

kivymd.fonts_path

Path to fonts directory.

kivymd.images_path

Path to images directory.

2.7.2 Submodules

Register KivyMD widgets to use without import

Register KivyMD widgets to use without import

API - `kivymd.factory_registers`

```
kivymd.factory_registers.r
```

Material Resources

API - `kivymd.material_resources`

```
kivymd.material_resources.dp
kivymd.material_resources.DEVICE_IOS
kivymd.material_resources.DEVICE_TYPE = desktop
kivymd.material_resources.MAX_NAV_DRAWER_WIDTH
kivymd.material_resources.TOUCH_TARGET_HEIGHT
```

Theming Dynamic Text

Two implementations. The first is based on color brightness obtained from- <https://www.w3.org/TR/AERT#color-contrast> The second is based on relative luminance calculation for sRGB obtained from- <https://www.w3.org/TR/2008/REC-WCAG20-20081211/#relativeluminancedef> and contrast ratio calculation obtained from- <https://www.w3.org/TR/2008/REC-WCAG20-20081211/#contrast-ratiodef>

Preliminary testing suggests color brightness more closely matches the *Material Design spec* suggested text colors, but the alternative implementation is both newer and the current ‘correct’ recommendation, so is included here as an option.

API - `kivymd.theming_dynamic_text`

```
kivymd.theming_dynamic_text.get_contrast_text_color(color, use_color_brightness=True)
kivymd.theming_dynamic_text.color
```

kivymd.effects

API - kivymd.effects

Submodules

RouletteScrollEffect

This is a subclass of `kivy.effects.ScrollEffect` that simulates the motion of a roulette, or a notched wheel (think Wheel of Fortune). It is primarily designed for emulating the effect of the iOS and android date pickers.

Usage

Here's an example of using `RouletteScrollEffect` for a `kivy.uix.scrollviewScrollView`:

```
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.scrollview import ScrollView

# Preparing a `GridLayout` inside a `ScrollView`.
layout = GridLayout(cols=1, padding=10, size_hint=(None, None), width=500)
layout.bind(minimum_height=layout.setter('height'))

for i in range(30):
    btn = Button(text=str(i), size=(480, 40), size_hint=(None, None))
    layout.add_widget(btn)

root = ScrollView(
    size_hint=(None, None),
    size=(500, 320),
    pos_hint={'center_x': .5, 'center_y': .5},
    do_scroll_x=False,
)
root.add_widget(layout)

# Preparation complete. Now add the new scroll effect.
root.effect_y = RouletteScrollEffect(anchor=20, interval=40)
runTouchApp(root)
```

Here the `ScrollView` scrolls through a series of buttons with height 40. We then attached a `RouletteScrollEffect` with interval 40, corresponding to the button heights. This allows the scrolling to stop at the same offset no matter where it stops. The `RouletteScrollEffect.anchor` adjusts this offset.

Customizations

Other settings that can be played with include:

`RouletteScrollEffect.pull_duration`, `RouletteScrollEffect.coasting_alpha`,
`RouletteScrollEffect.pull_back_velocity`, and `RouletteScrollEffect.terminal_velocity`.

See their module documentations for details.

`RouletteScrollEffect` has one event `on_coasted_to_stop` that is fired when the roulette stops, “making a selection”. It can be listened to for handling or cleaning up choice making.

API - kivymd.effects.roulettescroll

class kivymd.effects.roulettescroll.RouletteScrollEffect(kwargs)**

This is a subclass of `kivy.effects.ScrollEffect` that simulates the motion of a roulette, or a notched wheel (think Wheel of Fortune). It is primarily designed for emulating the effect of the iOS and android date pickers.

New in version 0.104.2.

drag_threshold

Overrides `ScrollEffect.drag_threshold` to abolish drag threshold.

Note: If using this with a Roulette or other Tickline subclasses, what matters is `Tickline.drag_threshold`, which is passed to this attribute in the end.

`drag_threshold` is an `NumericProperty` and defaults to `0`.

min

max

interval

The interval of the values of the “roulette”.

`interval` is an `NumericProperty` and defaults to `50`.

anchor

One of the valid stopping values.

`anchor` is an `NumericProperty` and defaults to `0`.

pull_duration

When movement slows around a stopping value, an animation is used to pull it toward the nearest value. `pull_duration` is the duration used for such an animation.

`pull_duration` is an `NumericProperty` and defaults to `0.2`.

coasting_alpha

When within `coasting_alpha * interval` of the next notch and velocity is below `terminal_velocity`, coasting begins and will end on the next notch.

`coasting_alpha` is an `NumericProperty` and defaults to `0.5`.

pull_back_velocity

The velocity below which the scroll value will be drawn to the `nearest` notch instead of the `next` notch in the direction travelled.

`pull_back_velocity` is an `NumericProperty` and defaults to `50sp`.

terminal_velocity

If velocity falls between `pull_back_velocity` and `terminal_velocity` then the movement will start to coast to the next coming stopping value.

`terminal_velocity` is computed from a set formula given `interval`, `coasting_alpha`, `pull_duration`, and `friction`. Setting `terminal_velocity` has the effect of setting `pull_duration`.

get_term_vel(self)**set_term_vel(self, val)****start(self, val, t=None)**

Start the movement.

Parameters

val: float or int Value of the movement

t: float, defaults to None Time when the movement happen. If no time is set, it will use `time.time()`

on_notch(self, *args)**nearest_notch(self, *args)****next_notch(self, *args)****near_notch(self, d=0.01)****near_next_notch(self, d=None)****update_velocity(self, dt)**

(internal) Update the velocity according to the frametime and friction.

on_coasted_to_stop(self, *args)

This event fires when the roulette has stopped, *making a selection*.

Stiff Scroll Effect

An Effect to be used with ScrollView to prevent scrolling beyond the bounds, but politely.

A ScrollView constructed with StiffScrollEffect, eg. `ScrollView(effect_cls=StiffScrollEffect)`, will get harder to scroll as you get nearer to its edges. You can scroll all the way to the edge if you want to, but it will take more finger-movement than usual.

Unlike DampedScrollEffect, it is impossible to overscroll with StiffScrollEffect. That means you cannot push the contents of the ScrollView far enough to see what's beneath them. This is appropriate if the ScrollView contains, eg., a background image, like a desktop wallpaper. Overscrolling may give the impression that there is some reason to overscroll, even if just to take a peek beneath, and that impression may be misleading.

StiffScrollEffect was written by Zachary Spector. His other stuff is at: <https://github.com/LogicalDash/> He can be reached, and possibly hired, at: zacharyspector@gmail.com

API - kivymd.effects.stiffscroll

```
class kivymd.effects.stiffscroll.StiffScrollView(**kwargs)
```

Kinetic effect class. See module documentation for more information.

drag_threshold

Minimum distance to travel before the movement is considered as a drag.

drag_threshold is an `NumericProperty` and defaults to '20sp'.

min

Minimum boundary to stop the scrolling at.

min is an `NumericProperty` and defaults to 0.

max

Maximum boundary to stop the scrolling at.

max is an `NumericProperty` and defaults to 0.

max_friction

How hard should it be to scroll, at the worst?

max_friction is an `NumericProperty` and defaults to 1.

body

Proportion of the range in which you can scroll unimpeded.

body is an `NumericProperty` and defaults to 0.7.

scroll

Computed value for scrolling

scroll is an `NumericProperty` and defaults to 0.0.

transition_min

The AnimationTransition function to use when adjusting the friction near the minimum end of the effect.

transition_min is an `ObjectProperty` and defaults to `kivy.animation.AnimationTransition`.

transition_max

The AnimationTransition function to use when adjusting the friction near the maximum end of the effect.

transition_max is an `ObjectProperty` and defaults to `kivy.animation.AnimationTransition`.

target_widget

The widget to apply the effect to.

target_widget is an `ObjectProperty` and defaults to None.

displacement

The absolute distance moved in either direction.

displacement is an `NumericProperty` and defaults to 0.

update_velocity(self, dt)

Before actually updating my velocity, meddle with `self.friction` to make it appropriate to where I'm at, currently.

on_value(self, *args)

Prevent moving beyond my bounds, and update `self.scroll`

start(self, val, t=None)

Start movement with `self.friction = self.base_friction`

update(*self, val, t=None*)

Reduce the impact of whatever change has been made to me, in proportion with my current friction.

stop(*self, val, t=None*)

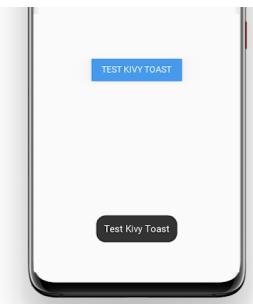
Work out whether I've been flung.

kivymd.toast

API - kivymd.toast

Submodules

Toast for Android device



API - kivymd.toast.androidtoast

Submodules

AndroidToast

Native implementation of toast for Android devices.

```
# Will be automatically used native implementation of the toast
# if your application is running on an Android device.
# Otherwise, will be used toast implementation
# from the kivymd/toast/kivytoast package.

from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager

from kivymd.toast import toast
from kivymd.app import MDApp

KV = '''
MDScreen:
```

(continues on next page)

(continued from previous page)

```

MDFlatButton:
    text: "My Toast"
    pos_hint: {"center_x": .5, "center_y": .5}
    on_press: app.show_toast()
...

class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

    def show_toast(self):
        toast("Hello World", True, 80, 200, 0)

Test().run()

```

API - kivymd.toast.androidtoast.androidtoast

`kivymd.toast.androidtoast.androidtoast`(*text*, *length_long=False*, *gravity=0*, *y=0*, *x=0*)
Displays a toast.

Parameters

- **length_long** – the amount of time (in seconds) that the toast is visible on the screen.
- **text** – text to be displayed in the toast;
- **short_duration** – duration of the toast, if *True* the toast will last 2.3s but if it is *False* the toast will last 3.9s;
- **gravity** – refers to the toast position, if it is 80 the toast will be shown below, if it is 40 the toast will be displayed above;
- **y** – refers to the vertical position of the toast;
- **x** – refers to the horizontal position of the toast;

Important: if only the text value is specified and the value of the *gravity*, *y*, *x* parameters is not specified, their values will be 0 which means that the toast will be shown in the center.

kivymd.toast.kivytoast

API - kivymd.toast.kivytoast

Submodules

KivyToast

Implementation of toasts for desktop.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.toast import toast

KV = '''
MDScreen:

    MDToolbar:
        title: 'Test Toast'
        pos_hint: {'top': 1}
        left_action_items: [['menu', lambda x: x]]

    MDRaisedButton:
        text: 'TEST KIVY TOAST'
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_toast()
'''

class Test(MDApp):
    def show_toast(self):
        """Displays a toast on the screen."""

        toast('Test Kivy Toast')

    def build(self):
        return Builder.load_string(KV)

Test().run()
```

API - `kivymd.toast.kivytoast.kivytoast`

```
class kivymd.toast.kivytoast.kivytoast.Toast(**kwargs)
    ModalView class. See module documentation for more information.
```

Events

`on_pre_open`: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

`on_open`: Fired when the ModalView is opened.

`on_pre_dismiss`: Fired before the ModalView is closed.

`on_dismiss`: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events `on_pre_open` and `on_pre_dismiss`.

Changed in version 2.0.0: Added property ‘`overlay_color`’.

duration

The amount of time (in seconds) that the toast is visible on the screen.

duration is an `NumericProperty` and defaults to 2.5.

`label_check_texture_size(self, instance, texture_size)`

`toast(self, text_toast)`

`on_open(self)`

`fade_in(self)`

`fade_out(self, *args)`

`on_touch_down(self, touch)`

Receive a touch down event.

Parameters

`touch: MotionEvent class` Touch received. The touch is in parent coordinates. See `relativelayout` for a discussion on coordinate systems.

Returns bool If True, the dispatching of the touch event will stop. If False, the event will continue to be dispatched to the rest of the widget tree.

`kivymd.toast.kivytoast.kivytoast.toast(text='', background=[0.2, 0.2, 0.2, 1], duration=2.5)`

Displays a toast.

Attr **duration** the amount of time (in seconds) that the toast is visible on the screen

Attr **background** color `rgba` in Kivy format

kivymd.tools

API - kivymd.tools

Submodules

kivymd.tools.packaging

API - kivymd.tools.packaging

Submodules

PyInstaller hooks

Add `hookspath=[kivymd.hooks_path]` to your .spec file.

Example of .spec file

```
# -*- mode: python ; coding: utf-8 -*-

import sys
import os

from kivy_deps import sdl2, glew

from kivymd import hooks_path as kivymd_hooks_path

path = os.path.abspath(".")

a = Analysis(
    ["main.py"],
    pathex=[path],
    hookspath=[kivymd_hooks_path],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=None,
    noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=None)

exe = EXE(
    pyz,
    a.scripts,
    a.binaries,
    a.zipfiles,
    a.datas,
    *[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins)],
    debug=False,
    strip=False,
    upx=True,
    name="app_name",
    console=True,
)
```

API - `kivymd.tools.packaging.pyinstaller`

`kivymd.tools.packaging.pyinstaller.hooks_path`

Path to hook directory to use with PyInstaller. See `kivymd.tools.packaging.pyinstaller` for more information.

`kivymd.tools.packaging.pyinstaller.get_hook_dirs()`

`kivymd.tools.packaging.pyinstaller.get_pyinstaller_tests()`

Submodules

PyInstaller hook for KivyMD

Adds fonts and images to package.

All modules from uix directory are added by Kivy hook.

API - kivymd.tools.packaging.pyinstaller.hook-kivymd

```
kivymd.tools.packaging.pyinstaller.hook-kivymd.datas = [None, None]
```

kivymd.tools.release

API - kivymd.tools.release

Submodules

kivymd.tools.release.argument_parser

API - kivymd.tools.release.argument_parser

```
class kivymd.tools.release.argument_parser.ArgumentParserWithHelp(prog=None, usage=None,
                                                               description=None,
                                                               epilog=None, parents=[], formatter_class=HelpFormatter,
                                                               prefix_chars='-', fromfile_prefix_chars=None,
                                                               argument_default=None, conflict_handler='error',
                                                               add_help=True, allow_abbrev=True)
```

Object for parsing command line strings into Python objects.

Keyword Arguments:

- prog – The name of the program (default: sys.argv[0])
- usage – A usage message (default: auto-generated from arguments)
- description – A description of what the program does
- epilog – Text following the argument descriptions
- parents – Parsers whose arguments should be copied into this one
- formatter_class – HelpFormatter class for printing help messages
- prefix_chars – Characters that prefix optional arguments
- **fromfile_prefix_chars – Characters that prefix files containing additional arguments**
- argument_default – The default value for all arguments

- conflict_handler – String indicating how to handle conflicts
- add_help – Add a -h/-help option
- allow_abbrev – Allow long options to be abbreviated unambiguously

parse_args(self, args=None, namespace=None)

error(self, message)

error(message: string)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

format_help(self)

kivymd.tools.release.git_commands

API - kivymd.tools.release.git_commands

kivymd.tools.release.git_commands.command(cmd: list, capture_output: bool = False) → str

Run system command.

kivymd.tools.release.git_commands.get_previous_version() → str

Returns latest tag in git.

kivymd.tools.release.git_commands.git_clean(ask: bool = True)

Clean git repository from untracked and changed files.

kivymd.tools.release.git_commands.git_commit(message: str, allow_error: bool = False, add_files: list = None)

Make commit.

kivymd.tools.release.git_commands.git_tag(name: str)

Create tag.

kivymd.tools.release.git_commands.git_push(branches_to_push: list, ask: bool = True, push: bool = False)

Push all changes.

Script to make release

Run this script before release (before deploying).

What this script does:

- Undo all local changes in repository
- Update version in `__init__.py`, `README.md`
- Format files
- Rename file “unreleased.rst” to version, add to index.rst
- Commit “Version ...”
- Create tag
- Add `unreleased.rst` to Changelog, add to `index.rst`

- Commit
- Git push

API - `kivymd.tools.release.make_release`

`kivymd.tools.release.make_release.run_pre_commit()`

Run pre-commit.

`kivymd.tools.release.make_release.replace_in_file(pattern, repl, file)`

Replace one *pattern* match to *repl* in file *file*.

`kivymd.tools.release.make_release.update_init_py(version, is_release, test: bool = False)`

Change version in *kivymd/_init_.py*.

`kivymd.tools.release.make_release.update_readme(previous_version, version, test: bool = False)`

Change version in *README.md*.

`kivymd.tools.release.make_release.move_changelog(index_file, unreleased_file, previous_version, version_file, version, test: bool = False)`

Edit unreleased.rst and rename to <version>.rst.

`kivymd.tools.release.make_release.create_unreleased_changelog(index_file, unreleased_file, version, ask: bool = True, test: bool = False)`

Create unreleased.rst by template.

`kivymd.tools.release.make_release.main()`

`kivymd.tools.release.make_release.create_argument_parser()`

Tool for updating Iconic font

Downloads archive from <https://github.com/Templarian/MaterialDesign-Webfont> and updates font file with icon_definitions.

API - `kivymd.tools.release.update_icons`

`kivymd.tools.release.update_icons.kivymd_path`

`kivymd.tools.release.update_icons.font_path`

`kivymd.tools.release.update_icons.icon_definitions_path`

`kivymd.tools.release.update_icons.font_version = master`

`kivymd.tools.release.update_icons.url`

`kivymd.tools.release.update_icons.temp_path`

`kivymd.tools.release.update_icons.temp_repo_path`

`kivymd.tools.release.update_icons.temp_font_path`

`kivymd.tools.release.update_icons.temp_preview_path`

`kivymd.tools.release.update_icons.re_icons_json`

```
kivymd.tools.release.update_icons.re_additional_icons
kivymd.tools.release.update_icons.re_version
kivymd.tools.release.update_icons.re_quote_keys
kivymd.tools.release.update_icons.re_icon_definitions
kivymd.tools.release.update_icons.re_version_in_file
kivymd.tools.release.update_icons.download_file(url, path)
kivymd.tools.release.update_icons.unzip_archive(archive_path, dir_path)
kivymd.tools.release.update_icons.get_icons_list()
kivymd.tools.release.update_icons.make_icon_definitions(icons)
kivymd.tools.release.update_icons.export_icon_definitions(icon_definitions, version)
kivymd.tools.release.update_icons.update_icons(make_commit: bool = False)
kivymd.tools.release.update_icons.main()
```

kivymd.uix

API - kivymd.uix

```
class kivymd.uix.MDAdaptiveWidget(**kwargs)
```

Common base class for rectangular and circular elevation behavior.

adaptive_height

If *True*, the following properties will be applied to the widget:

```
size_hint_y: None
height: self.minimum_height
```

adaptive_height is an BooleanProperty and defaults to *False*.

adaptive_width

If *True*, the following properties will be applied to the widget:

```
size_hint_x: None
width: self.minimum_width
```

adaptive_width is an BooleanProperty and defaults to *False*.

adaptive_size

If *True*, the following properties will be applied to the widget:

```
size_hint: None, None
size: self.minimum_size
```

adaptive_size is an BooleanProperty and defaults to *False*.

```
on_adaptive_height(self, instance, value)
```

```
on_adaptive_width(self, instance, value)
```

```
on_adaptive_size(self, instance, value)
```

Submodules

Behaviors

Modules and classes implementing various behaviors for buttons etc.

API - `kivymd.uix.behaviors`

Submodules

`kivymd.utils`

API - `kivymd.utils`

Submodules

`asynckivy`

Copyright (c) 2019 Nattōsai Mitō

GitHub - <https://github.com/gottadiveintopython>

GitHub Gist - <https://gist.github.com/gottadiveintopython/5f4a775849f9277081c396de65dc57c1>

API - `kivymd.utils.asynckivy`

```
kivymd.utils.asynckivy.start(coro)
kivymd.utils.asynckivy.sleep(duration)
class kivymd.utils.asynckivy.event(ed, name)  
  

bind(self, step_coro)
callback(self, *args, **kwargs)
```

Fit Image

Feature to automatically crop a *Kivy* image to fit your layout Write by Benedikt Zwölfer

Referene - <https://gist.github.com/benni12er/95a45eb168fc33a4fcd2d545af692dad>

Example:

```
MDBoxLayout:  
    size_hint_y: None  
    height: "200dp"  
    orientation: 'vertical'  
  
    FitImage:  
        size_hint_y: 3  
        source: 'images/img1.jpg'  
  
    FitImage:  
        size_hint_y: 1  
        source: 'images/img2.jpg'
```

Example with round corners:

```
from kivy.uix.modalview import ModalView
from kivy.lang import Builder

from kivymd import images_path
from kivymd.app import MDApp
from kivymd.uix.card import MDCard

Builder.load_string(
    """
<Card>:
    elevation: 10
    radius: [36, ]
```

(continues on next page)

(continued from previous page)

```
FitImage:  
    id: bg_image  
    source: "images/bg.png"  
    size_hint_y: .35  
    pos_hint: {"top": 1}  
    radius: 36, 36, 0, 0  
")  
  
class Card(MDCard):  
    pass  
  
class Example(MDApp):  
    def build(self):  
        modal = ModalView(  
            size_hint=(0.4, 0.8),  
            background=f"images_path]/transparent.png",  
            overlay_color=(0, 0, 0, 0),  
        )  
        modal.add_widget(Card())  
        modal.open()  
  
Example().run()
```

API - kivymd.utils.fitimage

```
class kivymd.utils.fitimage.FitImage(**kwargs)  
Box layout class. See module documentation for more information.
```

source

Filename/source of your image.

`source` is a `StringProperty` and defaults to None.

radius

Canvas radius.

```
# Top left corner slice.  
MDBBoxLayout:  
    md_bg_color: app.theme_cls.primary_color  
    radius: [25, 0, 0, 0]
```

`radius` is an `VariableListProperty` and defaults to [0, 0, 0, 0].

reload(self)

Monitor module

The Monitor module is a toolbar that shows the activity of your current application :

- FPS

API - `kivymd.utils.fpsmonitor`

```
class kivymd.utils.fpsmonitor.FpsMonitor(**kwargs)
```

Label class, see module documentation for more information.

Events

`on_ref_press` Fired when the user clicks on a word referenced with a `[ref]` tag in a text markup.

`updated_interval`

FPS refresh rate.

`start(self)`

`update_fps(self, *args)`

HotReloadViewer

Note: The `HotReloadViewer` class is based on the `KvViewerApp` class

`HotReloadViewer`, for KV-Viewer, is a simple tool allowing you to dynamically display a KV file, taking its changes into account (thanks to watchdog). The idea is to facilitate design using the KV language.

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
#:import KivyLexer kivy.extras.highlight.KivyLexer
#:import HotReloadViewer kivymd.utils.hot_reload_viewer.HotReloadViewer
```

BoxLayout :

```
CodeInput:
    lexer: KivyLexer()
    style_name: "native"
    on_text: app.update_kv_file(self.text)
    size_hint_x: .7
```

(continues on next page)

(continued from previous page)

```

HotReloadViewer:
    size_hint_x: .3
    path: app.path_to_kv_file
    errors: True
    errors_text_color: 1, 1, 0, 1
    errors_background_color: app.theme_cls.bg_dark
...
```
class Example(MDApp):
 path_to_kv_file = "kv_file.kv"

 def build(self):
 self.theme_cls.theme_style = "Dark"
 return Builder.load_string(KV)

 def update_kv_file(self, text):
 with open(self.path_to_kv_file, "w") as kv_file:
 kv_file.write(text)

Example().run()
```

```

This will display the test.kv and automatically update the display when the file changes.

This scripts uses watchdog to listen for file changes. To install watchdog.

```
pip install watchdog
```

API - kivymd.utils.hot_reload_viewer

```
class kivymd.utils.hot_reload_viewer.HotReloadErrorText(**kwargs)
ScrollView class. See module documentation for more information.
```

Events

on_scroll_start Generic event fired when scrolling starts from touch.

on_scroll_move Generic event fired when scrolling move from touch.

on_scroll_stop Generic event fired when scrolling stops from touch.

Changed in version 1.9.0: *on_scroll_start*, *on_scroll_move* and *on_scroll_stop* events are now dispatched when scrolling to handle nested ScrollViews.

Changed in version 1.7.0: *auto_scroll*, *scroll_friction*, *scroll_moves*, *scroll_stoptime*' has been deprecated, use :attr:`effect_cls` instead.

text

Text errors.

text is an *StringProperty* and defaults to ‘’.

errors_text_color

Error text color.

```
    errors_text_color is an ColorProperty and defaults to None.  
class kivymd.utils.hot_reload_viewer.HotReloadHandler(callback, target, **kwargs)  
  
    on_any_event(self, event)  
class kivymd.utils.hot_reload_viewer.HotReloadViewer(**kwargs)
```

Events

on_error Called when an error occurs in the KV-file that the user is editing.

path

Path to KV file.

path is an StringProperty and defaults to ‘‘.

errors

Show errors while editing KV-file.

errors is an BooleanProperty and defaults to *False*.

errors_background_color

Error background color.

errors_background_color is an ColorProperty and defaults to *None*.

errors_text_color

Error text color.

errors_text_color is an ColorProperty and defaults to *None*.

update(self, *args)

Updates and displays the KV-file that the user edits.

show_error(self, error)

Displays text with a current error.

on_error(self, *args)

Called when an error occurs in the KV-file that the user is editing.

on_errors_text_color(self, instance, value)

on_path(self, instance, value)

CHAPTER
THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

kivymd, 322
kivymd.app, 20
kivymd.color_definitions, 22
kivymd.effects, 324
kivymd.effects.roulettescroll, 324
kivymd.effects.stiffscroll, 326
kivymd.factory_registers, 323
kivymd.font_definitions, 27
kivymd.icon_definitions, 25
kivymd.material_resources, 323
kivymd.theming, 6
kivymd.theming_dynamic_text, 323
kivymd.toast, 328
kivymd.toast.androidtoast, 328
kivymd.toast.androidtoast.androidtoast, 328
kivymd.toast.kivytoast, 329
kivymd.toast.kivytoast.kivytoast, 329
kivymd.tools, 331
kivymd.tools.packaging, 331
kivymd.tools.packaging.pyinstaller, 331
kivymd.tools.packaging.pyinstaller.hook-kivymd, 333
kivymd.tools.release, 333
kivymd.tools.release.argument_parser, 333
kivymd.tools.release.git_commands, 334
kivymd.tools.release.make_release, 334
kivymd.tools.release.update_icons, 335
kivymd.uix, 336
kivymd.uix.backdrop, 77
kivymd.uix.banner, 274
kivymd.uix.behaviors, 337
kivymd.uix.behaviors.backgroundcolor_behavior, 311
kivymd.uix.behaviors.elevation, 286
kivymd.uix.behaviors.focus_behavior, 300
kivymd.uix.behaviors.hover_behavior, 284
kivymd.uix.behaviors.magic_behavior, 307
kivymd.uix.behaviors.ripple_behavior, 302
kivymd.uix.behaviors.toggle_behavior, 305
kivymd.uix.behaviors.touch_behavior, 309
kivymd.uix.bottomnavigation, 268

kivymd.uix.bottomsheet, 211
kivymd.uix.boxlayout, 237
kivymd.uix.button, 219
kivymd.uix.card, 197
kivymd.uix.carousel, 265
kivymd.uix.chip, 81
kivymd.uix.circularlayout, 127
kivymd.uix.datatables, 47
kivymd.uix.dialog, 179
kivymd.uix.dropdownitem, 89
kivymd.uix.expansionpanel, 261
kivymd.uix.filemanager, 122
kivymd.uix.floatlayout, 91
kivymd.uix.gridlayout, 85
kivymd.uix.imagelist, 189
kivymd.uix.label, 161
kivymd.uix.list, 92
kivymd.uix.menu, 29
kivymd.uix.navigationdrawer, 131
kivymd.uix.navigationrail, 59
kivymd.uix.picker, 140
kivymd.uix.progressbar, 118
kivymd.uix.refreshlayout, 129
kivymd.uix.relativelayout, 237
kivymd.uix.screen, 28
kivymd.uix.selection, 278
kivymd.uix.selectioncontrol, 192
kivymd.uix.slider, 245
kivymd.uix.snackbar, 239
kivymd.uix.spinner, 234
kivymd.uix.stacklayout, 267
kivymd.uix.swiper, 41
kivymd.uix.tab, 104
kivymd.uix.taptargetview, 166
kivymd.uix.textfield, 247
kivymd.uix.toolbar, 67
kivymd.uix.tooltip, 86
kivymd.utils, 337
kivymd.utils.asynckivy, 337
kivymd.utils.fitimage, 337
kivymd.utils.fpsmonitor, 341
kivymd.utils.hot_reload_viewer, 341

INDEX

A

a (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute*), 311
accent_color (*kivymd.theming.ThemeManager attribute*), 13
accent_color (*kivymd.uix.picker.BaseDialogPicker attribute*), 148
accent_dark (*kivymd.theming.ThemeManager attribute*), 13
accent_dark_hue (*kivymd.theming.ThemeManager attribute*), 13
accent_hue (*kivymd.theming.ThemeManager attribute*), 13
accent_light (*kivymd.theming.ThemeManager attribute*), 13
accent_light_hue (*kivymd.theming.ThemeManager attribute*), 13
accent_palette (*kivymd.theming.ThemeManager attribute*), 12
action_color_button
 (*kivymd.uix.navigationrail.MDNavigationRail attribute*), 65
action_icon_button (*kivymd.uix.navigationrail.MDNavigationRail attribute*), 65
action_text_button (*kivymd.uix.navigationrail.MDNavigationRail attribute*), 65
active (*kivymd.uix.selectioncontrol.MDCheckbox attribute*), 196
active (*kivymd.uix.selectioncontrol.MDSwitch attribute*), 197
active (*kivymd.uix.slider.MDSlider attribute*), 246
active (*kivymd.uix.spinner.MDSpinner attribute*), 236
active_line (*kivymd.uix.textfield.MDTextField attribute*), 258
adaptive_height (*kivymd.uix.MDAdaptiveWidget attribute*), 336
adaptive_size (*kivymd.uix.MDAdaptiveWidget attribute*), 336
adaptive_width (*kivymd.uix.MDAdaptiveWidget attribute*), 336
add_actions_buttons()
 (*kivymd.uix.banner.MDBanner method*), 278
add_banner_to_container()
 (*kivymd.uix.banner.MDBanner method*), 278
add_item() (*kivymd.uix.bottomsheet.MDGridBottomSheet method*), 219
add_item() (*kivymd.uix.bottomsheet.MDListBottomSheet method*), 218
add_scrim() (*kivymd.uix.navigationdrawer.MDNavigationLayout method*), 137
add_widget() (*kivymd.uix.backdrop.MDBackdrop method*), 80
add_widget() (*kivymd.uix.bottomnavigation.MDBottomNavigation method*), 273
add_widget() (*kivymd.uix.bottomsheet.MDBottomSheet method*), 217
add_widget() (*kivymd.uix.card.MDCardSwipe method*), 209
add_widget() (*kivymd.uix.chip.MDChooseChip method*), 84
add_widget() (*kivymd.uix.expansionpanel.MDExpansionPanel method*), 264
add_widget() (*kivymd.uix.list.MDList method*), 102
add_widget() (*kivymd.uix.navigationdrawer.MDNavigationRail method*), 137
add_widget() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
add_widget() (*kivymd.uix.selection.MDSelectionList method*), 283
add_widget() (*kivymd.uix.swiper.MDSwiper method*), 45
add_widget() (*kivymd.uix.tab.MDTabs method*), 117
add_widget() (*kivymd.uix.toolbar.MDBottomAppBar method*), 76
adjust_tooltip_position()
 (*kivymd.uix.tooltip.MDTooltip method*), 89
allow_stretch (*kivymd.uix.tab.MDTabs attribute*), 116
am_pm (*kivymd.uix.spinner.MDTimePicker attribute*), 160
am_pm_border_width (*kivymd.uix.spinner.MDTimePicker attribute*), 159
am_pm_radius (*kivymd.uix.spinner.MDTimePicker attribute*), 159

anchor (*kivymd.effects.roulettescroll.RouletteScrollEffect attribute*), 325
anchor (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 231
anchor (*kivymd.uix.card.MDCardSwipe attribute*), 208
anchor (*kivymd.uix.navigationdrawer.MDNavigationDrawer attribute*), 138
anchor_title (*kivymd.uix.toolbar.MDToolbar attribute*), 75
angle (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundBehavior attribute*), 312
angle (*kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute*), 295
anim_color_active() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
anim_color_normal() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
anim_complete() (*kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute*), 305
anim_duration (*kivymd.uix.tab.MDTabs attribute*), 115
anim_rect() (*kivymd.uix.textfield.MDTextFieldRect method*), 256
anim_threshold (*kivymd.uix.tab.MDTabs attribute*), 115
animation (*kivymd.uix.bottomsheetMDBottomSheet attribute*), 216
animation_display_banner() (*kivymd.uix.banner.MDBanner method*), 278
animation_duration (*kivymd.uix.picker.MDTimePicker attribute*), 160
animation_label() (*kivymd.uix.button.MDTextButton method*), 230
animation_tooltip_show() (*kivymd.uix.tooltip.MDTooltip method*), 89
animation_transition (*kivymd.uix.picker.MDTimePicker attribute*), 160
animtion_icon_close() (*kivymd.uix.backdrop.MDBackdrop method*), 80
animtion_icon_menu() (*kivymd.uix.backdrop.MDBackdrop method*), 80
ArgumentParserWithHelp (*class in kivymd.tools.release.argument_parser*), 333

B

b (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute*), 311
back() (*kivymd.uix.filemanager.MDFileManager method*), 127
back_layer_color (*kivymd.uix.backdrop.MDBackdrop attribute*), 79
background (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundBehavior attribute*), 311
background (*kivymd.uix.bottomsheet.MDBottomSheet attribute*), 216
background_color (*kivymd.uix.datatables.MDDatatable attribute*), 58
background_color (*kivymd.uix.menu.MDDropdownMenu attribute*), 109
background_color (*kivymd.uix.tab.MDTabs attribute*), 116
background_down (*kivymd.uix.behaviors.toggle_behavior.MDToggleButton attribute*), 307
background_hue (*kivymd.uix.behaviors.backgroundcolor_behavior.SpecifyHueBehavior attribute*), 312
background_normal (*kivymd.uix.behaviors.toggle_behavior.MDToggleButton attribute*), 307
background_origin (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundOriginBehavior attribute*), 312
background_palette (*kivymd.uix.behaviors.backgroundcolor_behavior.SpecifyColorBehavior attribute*), 312
BackgroundColorBehavior (*class in kivymd.uix.behaviors.backgroundcolor_behavior*), 311
BaseDialogPicker (*class in kivymd.uix.picker*), 147
bg_color (*kivymd.uix.bottomsheet.MDBottomSheet attribute*), 216
bg_color_root_button (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
bg_color_stack_button (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
bg_dark (*kivymd.theming.ThemeManager attribute*), 15
bg_darkest (*kivymd.theming.ThemeManager attribute*), 14
bg_hint_color (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 233
bg_light (*kivymd.theming.ThemeManager attribute*), 15
bg_normal (*kivymd.theming.ThemeManager attribute*), 15
bind() (*kivymd.utils.asynckivy.event method*), 337
body (*kivymd.effects.stiffscroll.StiffScrollEffect attribute*), 327
border_margin (*kivymd.uix.menu.MDDropdownMenu attribute*), 39
box_color (*kivymd.uix.imagelist.SmartTile attribute*), 191
box_position (*kivymd.uix.imagelist.SmartTile attribute*), 191
buttons (*kivymd.uix.dialog.MDDialog attribute*), 182

C

callback (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 231
callback() (*kivymd.utils.asynckivy.event method*), 337
caller (*kivymd.uix.menu.MDDropdownMenu attribute*), 40
can_capitalize (*kivymd.uix.label.MDLabel attribute*), 165
cancelable (*kivymd.uix.taptargetview.MDTapTargetView attribute*), 177
caption (*kivymd.uix.bottomsheet.GridBottomSheetItem attribute*), 219
catching_duration (*kivymd.uix.progressbar.MDProgressBar attribute*), 122
catching_transition
 (*kivymd.uix.progressbar.MDProgressBar attribute*), 122
catching_up() (*kivymd.uix.progressbar.MDProgressBar method*), 122
change_month() (*kivymd.uix.picker.MDDatePicker method*), 159
check (*kivymd.uix.chip.MDChip attribute*), 84
check (*kivymd.uix.datatables.MDDDataTable attribute*), 55
check_determinate() (*kivymd.uix.spinner.MDSpinner method*), 236
check_font_styles() (*kivymd.uix.label.MDLabel method*), 165
check_open_panel() (*kivymd.uix.expansionpanel.MDEExpansionPanel method*), 264
check_position_caller()
 (*kivymd.uix.menu.MDDropdownMenu method*), 40
check_text() (*kivymd.uix.textfield.MDTextField method*), 259
checkbox_icon_down (*kivymd.uix.selectioncontrol.MDCheckbox attribute*), 196
checkbox_icon_normal
 (*kivymd.uix.selectioncontrol.MDCheckbox attribute*), 196
CheckboxLeftWidget (*class in kivymd.uix.list*), 104
circular_padding (*kivymd.uix.circularlayout.MDCircularLayout attribute*), 128
circular_radius (*kivymd.uix.circularlayout.MDCircularLayout attribute*), 128
CircularElevationBehavior (*class in kivymd.uix.behaviors.elevation*), 299
CircularRippleBehavior (*class in kivymd.uix.behaviors.ripple_behavior*), 305
clockwise (*kivymd.uix.circularlayout.MDCircularLayout attribute*), 128
close() (*kivymd.uix.backdrop.MDBackdrop method*), 80
close() (*kivymd.uix.filemanager.MDFileManager method*), 126
close() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
close_card() (*kivymd.uix.card.MDCardSwipe method*), 210
close_icon (*kivymd.uix.backdrop.MDBackdrop attribute*), 80
close_on_click (*kivymd.uix.navigationdrawer.MDNavigationDrawer attribute*), 138
close_panel() (*kivymd.uix.expansionpanel.MDEExpansionPanel method*), 264
close_stack() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 234
closing_time (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
closing_time (*kivymd.uix.expansionpanel.MDEExpansionPanel attribute*), 264
closing_time (*kivymd.uix.navigationdrawer.MDNavigationDrawer attribute*), 139
closing_time_button_rotation
 (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
closing_transition (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
closing_transition (*kivymd.uix.card.MDCardSwipe attribute*), 208
closing_transition (*kivymd.uix.expansionpanel.MDEExpansionPanel attribute*), 264
closing_transition (*kivymd.uix.navigationdrawer.MDNavigationDrawer attribute*), 139
closing_transition_button_rotation
 (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232
coasting_alpha (*kivymd.effects.roulettescroll.RouletteScrollEffect attribute*), 325
color (*in module kivymd.theming_dynamic_text*), 323
color (*kivymd.uix.button.MDTextButton attribute*), 230
color (*kivymd.uix.card.MDSeparator attribute*), 207
color (*kivymd.uix.chip.MDChip attribute*), 84
color (*kivymd.uix.progressbar.MDProgressBar attribute*), 121
color (*kivymd.uix.slider.MDSlider attribute*), 247
color (*kivymd.uix.spinner.MDSpinner attribute*), 236
color_active (*kivymd.uix.navigationrail.MDNavigationRail attribute*), 65
color_active (*kivymd.uix.textfield.MDTextFieldRound attribute*), 260
color_change_duration
 (*kivymd.uix.navigationrail.MDNavigationRail attribute*), 65
color_disabled (*kivymd.uix.button.MDTextButton attribute*), 230
color_icon_root_button
 (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 232

attribute), 233
color_icon_stack_button (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 233
color_mode (kivymd.uix.textfield.MDTextField attribute), 257
color_normal (kivymd.uix.navigationrail.MDNavigationRail attribute), 65
color_transition (kivymd.uix.navigationrail.MDNavigationRail attribute), 65
colors (in module kivymd.color_definitions), 22
column_data (kivymd.uix.datatables.MDDDataTable attribute), 51
command() (in module kivymd.tools.release.git_commands), 334
CommonElevationBehavior (class in kivymd.uix.behaviors.elevation), 294
CommonRipple (class in kivymd.uix.behaviors.ripple_behavior), 304
compare_date_range() (kivymd.uix.picker.MDDatePicker method), 158
complete_swipe() (kivymd.uix.card.MDCardSwipe method), 210
content (kivymd.uix.expansionpanel.MDExpansionPanel attribute), 264
content_cls (kivymd.uix.dialog.MDDialog attribute), 186
create_argument_parser() (in module kivymd.tools.release.make_release), 335
create_buttons() (kivymd.uix.dialog.MDDialog method), 188
create_clock() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 310
create_items() (kivymd.uix.dialog.MDDialog method), 188
create_pagination_menu() (kivymd.uix.datatables.MDDDataTable method), 59
create_unreleased_changelog() (in module kivymd.tools.release.make_release), 335
current (kivymd.uix.bottomnavigation.TabbedPanelBase attribute), 272
current_hint_text_color (kivymd.uix.textfield.MDTextField attribute), 258
current_item (kivymd.uix.dropdownitem.MDDropDownItem attribute), 91
current_path (kivymd.uix.filemanager.MDFileManager attribute), 126

D

data (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 231

datas (in module kivymd.tools.packaging.pyinstaller.hook_kivymd), 333
Date_range_text_error (kivymd.uix.picker.MDDatePicker attribute), 158
day (kivymd.uix.picker.MDDatePicker attribute), 157
default_tab (kivymd.uix.tab.MDTabs attribute), 115
degree_spacing (kivymd.uix.circularlayout.MDCircularLayout attribute), 128
delete_clock() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 310
delete_clock() (kivymd.uix.tooltip.MDTooltip method), 89
description_text (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
description_text_bold (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
description_text_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
description_text_size (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
detect_visible (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 286
determinate (kivymd.uix.spinner.MDSpinner attribute), 236
determinate_time (kivymd.uix.spinner.MDSpinner attribute), 236
DEVICE_IOS (in module kivymd.material_resources), 323
device_ios (kivymd.theming.ThemableBehavior attribute), 19
device_orientation (kivymd.theming.ThemeManager attribute), 17
DEVICE_TYPE (in module kivymd.material_resources), 323
disabled_color (kivymd.uix.selectioncontrol.MDCheckbox attribute), 196
disabled_hint_text_color (kivymd.theming.ThemeManager attribute), 16
dismiss() (kivymd.uix.bottomsheet.MDBottomSheet method), 217
dismiss() (kivymd.uix.menu.MDDropdownMenu method), 41
displacement (kivymd.effects.stiffscroll.StiffScrollEffect attribute), 327
display_tooltip() (kivymd.uix.tooltip.MDTooltip method), 89
divider_color (kivymd.theming.ThemeManager attribute), 16
do_animation_open_stack() (kivymd.uix.button.MDFloatingActionButtonSpeedDial method), 233

do_layout() (*kivymd.uix.circularlayout.MDCircularLayout*.event (class in *kivymd.utils.asynckivy*), 337
 method), 128

download_file() (in module *kivymd.tools.release.update_icons*), 336

dp (in module *kivymd.material_resources*), 323

drag_threshold (*kivymd.effects.roulettescroll.RouletteScrollEffect*.attribute), 325

drag_threshold (*kivymd.effects.stiffscroll.StiffScrollView*.attribute), 327

draw_shadow (*kivymd.uix.behaviors.elevation.CommonElevationBehavior*.attribute), 297

draw_shadow (*kivymd.uix.taptargetview.MDTapTargetView*.fade_out() (in *kivymd.toast.kivytoast.Toast* method), 331
 attribute), 177

duration (*kivymd.toast.kivytoast.kivytoast.Toast* attribute), 330

duration_closing (*kivymd.uix.bottomsheetMDBottomSheet*.FakeCircularElevationBehavior (class in *kivymd.uix.behaviors.elevation*), 300
 attribute), 216

duration_long_touch (*kivymd.uix.behaviors.touch_behavior.TouchBehavior*.attribute), 299
 attribute), 310

duration_opening (*kivymd.uix.bottomsheetMDBottomSheet*.FakeRectangularElevationBehavior (class in *kivymd.uix.behaviors.elevation*), 299
 attribute), 216

E

edit_padding_for_item() (*kivymd.uix.dialog.MDDialog* method), 188

elevation (*kivymd.uix.behaviors.elevation.CommonElevationBehavior*.attribute), 294

elevation (*kivymd.uix.card.MDCard* attribute), 208

elevation (*kivymd.uix.datatables.MDDataTable* attribute), 56

elevation (*kivymd.uix.tab.MDTabs* attribute), 116

elevation (*kivymd.uix.toolbar.NotchedBox* attribute), 74

enable_swiping (*kivymd.uix.navigationdrawer.MDNavigationDrawer*.attribute), 139

enter_point (*kivymd.uix.behaviors.hover_behavior.HoverBehavior*.attribute), 286

error (*kivymd.uix.textfield.MDTextField* attribute), 258

error() (*kivymd.tools.release.argument_parser.ArgumentParser*.withAttribute), 307
 method), 334

error_color (*kivymd.theming.ThemeManager* attribute), 17

error_color (*kivymd.uix.textfield.MDTextField* attribute), 258

errors (*kivymd.utils.hot_reload_viewer.HotReloadViewer*.attribute), 343

errors_background_color (*kivymd.utils.hot_reload_viewer.HotReloadViewer*.attribute), 343

errors_text_color (*kivymd.utils.hot_reload_viewer.HotReloadErrorText*.attribute), 259

errors_text_color (*kivymd.utils.hot_reload_viewer.HotReloadErrorText*.attribute), 342

errors_text_color (*kivymd.utils.hot_reload_viewer.HotReloadErrorText*.attribute), 343

F

exit_manager (*kivymd.uix.filemanager.MDFileManager*.attribute), 125

export_icon_definitions() (in module *kivymd.tools.release.update_icons*), 336

K

fade_in() (*kivymd.uix.behaviors.ripple_behavior.CommonRippleBehavior*.attribute), 305

fill_color (*kivymd.uix.textfield.MDTextField* attribute), 258

finish_ripple() (*kivymd.uix.behaviors.ripple_behavior.CommonRippleBehavior*.attribute), 305

first_widget (*kivymd.uix.bottomnavigation.MDBottomNavigation* attribute), 272

FitImage (class in *kivymd.utils.fitimage*), 340

fixed_tab_label_width (*kivymd.uix.tab.MDTabs* attribute), 116

focus_behavior (*kivymd.uix.behaviors.focus_behavior.FocusBehavior*.attribute), 302

focus_behavior (*kivymd.uix.card.MDCard* attribute), 208

focus_color (*kivymd.uix.behaviors.focus_behavior.FocusBehavior*.attribute), 302

FocusBehavior (class in *kivymd.uix.behaviors.focus_behavior*), 302

font_color_down (*kivymd.uix.behaviors.toggle_behavior.MDToggleButton*.attribute), 307

font_color_normal (*kivymd.uix.behaviors.toggle_behavior.MDToggleButton*.attribute), 307

font_name (*kivymd.uix.picker.BaseDialogPicker* attribute), 156

font_name (*kivymd.uix.tab.MDTabs* attribute), 116

font_name_helper_text (*kivymd.uix.textfield.MDTextField* attribute), 258

font_name_hint_text (*kivymd.uix.textfield.MDTextField* attribute), 259

font_name_max_length (*kivymd.uix.textfield.MDTextField* attribute), 259

font_text_color (*kivymd.uix.textfield.MDTextField* attribute), 259

font_text_color (*kivymd.uix.textfield.MDTextField* attribute), 335

font_text_color (*kivymd.tools.release.update_icons*), 335

`font_size (kivymd.uix.dropdownitem.MDDropDownItem attribute), 91`

`font_size (kivymd.uix.snackbar.Snackbar attribute), 244`

`font_size (kivymd.uix.textfield.MDTextField attribute), 258`

`font_style (kivymd.uix.imagelist.SmartTileWithLabel attribute), 192`

`font_style (kivymd.uix.label.MDLabel attribute), 165`

`font_styles (kivymd.theming.ThemeManager attribute), 17`

`font_version (in module kivymd.tools.release.update_icons), 335`

`fonts (in module kivymd.font_definitions), 28`

`fonts_path (in module kivymd), 322`

`force_shadow_pos() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 298`

`force_title_icon_mode (kivymd.uix.tab.MDTabs attribute), 117`

`format_help() (kivymd.tools.release.argument_parser.ArgumentParser method), 334`

`FpsMonitor (class in kivymd.utils.fpsmonitor), 341`

`front_layer_color (kivymd.uix.backdrop.MDBackdrop attribute), 79`

G

`g (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute), 311`

`generate_list_widgets_days() (kivymd.uix.picker.MDDatePicker method), 159`

`generate_list_widgets_years() (kivymd.uix.picker.MDDatePicker method), 159`

`get_access_string() (kivymd.uix.filemanager.MDFileManager method), 126`

`get_angle() (kivymd.uix.circularlayout.MDCircularLayout method), 128`

`get_color_instruction() (kivymd.uix.textfield.MDTextFieldRect method), 256`

`get_content() (kivymd.uix.filemanager.MDFileManager method), 126`

`get_contrast_text_color() (in module kivymd.theming_dynamic_text), 323`

`get_current_index() (kivymd.uix.swiper.MDSwiper method), 46`

`get_current_item() (kivymd.uix.swiper.MDSwiper method), 46`

`get_date_range() (kivymd.uix.picker.MDDatePicker method), 159`

`get_dist_from_side() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 139`

`get_field() (kivymd.uix.picker.MDDatePicker method), 159`

`get_hook_dirs() (in module kivymd.tools.packaging.pyinstaller), 332`

`get_icons_list() (in module kivymd.tools.release.update_icons), 336`

`get_items() (kivymd.uix.swiper.MDSwiper method), 46`

`get_normal_height() (kivymd.uix.dialog.MDDialog method), 188`

`get_previous_version() (in module kivymd.tools.release.git_commands), 334`

`get_pyinstaller_tests() (in module kivymd.tools.packaging.pyinstaller), 332`

`get_rect_instruction()`

`get_rect_instruction() (kivymd.uix.textfield.MDTextFieldRect method), 256`

`get_row_checks() (kivymd.uix.datatables.MDDDataTable method), 59`

`get_selected_list_items() (kivymd.uix.selection.MDSelectionList method), 284`

`get_slides() (kivymd.uix.tab.MDTabs method), 117`

`get_state() (kivymd.uix.expansionpanel.MDExpansionPanel method), 264`

`get_state() (kivymd.uix.picker.MDTimePicker method), 160`

`get_tab_list() (kivymd.uix.tab.MDTabs method), 117`

`get_term_vel() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326`

`git_clean() (in module kivymd.tools.release.git_commands), 334`

`git_commit() (in module kivymd.tools.release.git_commands), 334`

`git_push() (in module kivymd.tools.release.git_commands), 334`

`git_tag() (in module kivymd.tools.release.git_commands), 334`

`GridBottomSheetItem (class in kivymd.uix.bottomsheet), 218`

`grow() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 309`

H

`hard_shadow_cl (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 297`

`hard_shadow_offset (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 297`

`hard_shadow_pos (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 297`

`hard_shadow_size (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 297`

hard_shadow_texture (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 297

header (kivymd.uix.backdrop.MDBackdrop attribute), 80

header (kivymd.uix.bottomnavigation.MDBottomNavigation icon (kivymd.uix.button.MDFloatingActionButton attribute), 230

header_cls (kivymd.uix.menu.MDDropdownMenu attribute), 39

header_text (kivymd.uix.backdrop.MDBackdrop attribute), 80

helper_text (kivymd.uix.textfield.MDTextField attribute), 257

helper_text_mode (kivymd.uix.textfield.MDTextField attribute), 257

hide() (kivymd.uix.bannerMDBanner method), 278

hide_anim_spinner() (kivymd.uix.refreshlayout.RefreshSpinner method), 131

hint (kivymd.uix.slider.MDSlider attribute), 246

hint_animation (kivymd.uix.button.MDFloatingActionButton attribute), 233

hint_bg_color (kivymd.uix.slider.MDSlider attribute), 246

hint_radius (kivymd.uix.slider.MDSlider attribute), 247

hint_text_color (kivymd.uix.slider.MDSlider attribute), 246

hooks_path (in module kivymd.tools.packaging.pyinstaller), 332

hor_growth (kivymd.uix.menu.MDDropdownMenu attribute), 39

horizontal_margins (kivymd.theming.ThemeManager attribute), 17

HotReloadErrorText (class in kivymd.utils.hot_reload_viewer), 342

HotReloadHandler (class in kivymd.utils.hot_reload_viewer), 343

HotReloadViewer (class in kivymd.utils.hot_reload_viewer), 343

hour (kivymd.uix.picker.MDTimePicker attribute), 159

hour_radius (kivymd.uix.picker.MDTimePicker attribute), 159

hover_bg (kivymd.uix.navigationrail.MDNavigationRail attribute), 62

hover_visible (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 286

HoverBehavior (class in module kivymd.uix.behaviors.hover_behavior), 286

hue (in module kivymd.color_definitions), 24

|

icon (kivymd.uix.bannerMDBanner attribute), 277

icon (kivymd.uix.bottonnavigation.MDTab attribute), 272

icon (kivymd.uix.button.MDFloatingActionButton attribute), 231

icon (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 231

icon (kivymd.uix.button.MDIconButton attribute), 229

icon (kivymd.uix.button.MDRectangleFlatIconButton attribute), 228

icon (kivymd.uix.button.MDRoundFlatButton attribute), 228

icon (kivymd.uix.chip.MDChip attribute), 83

icon (kivymd.uix.expansionpanel.MDExpansionPanel attribute), 264

icon (kivymd.uix.filemanager.MDFFileManager attribute), 125

icon (kivymd.uix.label.MDIcon attribute), 165

icon (kivymd.uix.selection.MDSelectionList attribute), 283

icon (kivymd.uix.tab.MDTabsBase attribute), 114

icon (kivymd.uix.toolbar.MDToolbar attribute), 75

icon_bg_color (kivymd.uix.selection.MDSelectionList attribute), 283

icon_check_color (kivymd.uix.selection.MDSelectionList attribute), 283

icon_color (kivymd.theming.ThemeManager attribute), 16

icon_color (kivymd.uix.button.MDRectangleFlatButton attribute), 228

icon_color (kivymd.uix.button.MDRoundFlatButton attribute), 228

icon_color (kivymd.uix.chip.MDChip attribute), 84

icon_color (kivymd.uix.toolbar.MDToolbar attribute), 75

icon_definitions_path (in module kivymd.tools.release.update_icons), 335

icon_folder (kivymd.uix.filemanager.MDFFileManager attribute), 125

icon_left (kivymd.uix.textfield.MDTextFieldRound attribute), 260

icon_left_color (kivymd.uix.textfield.MDTextFieldRound attribute), 260

icon_pos (kivymd.uix.selection.MDSelectionList attribute), 283

icon_right (kivymd.uix.textfield.MDTextFieldRound attribute), 260

icon_right_color (kivymd.uix.textfield.MDTextFieldRound attribute), 258

icon_right_color (kivymd.uix.textfield.MDTextFieldRound attribute), 260

icon_size (kivymd.uix.bottomsheet.GridBottomSheetItem attribute), 219

icon_title (*kivymd.uix.navigationrail.MDNavigationRail* attribute), 64

IconLeftWidget (*class in kivymd.uix.list*), 104

IconRightWidget (*class in kivymd.uix.list*), 104

ILeftBodyTouch (*class in kivymd.uix.list*), 102

ImageLeftWidget (*class in kivymd.uix.list*), 104

ImageRightWidget (*class in kivymd.uix.list*), 104

images_path (*in module kivymd*), 322

indicator_color (*kivymd.uix.tab.MDTabs* attribute), 116

input_field_background_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 154

input_field_text_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 155

interval (*kivymd.effects.roulettescroll.RouletteScrollEffect* attribute), 325

IRightBodyTouch (*class in kivymd.uix.list*), 103

item_switch() (*kivymd.uix.navigationrail.MDNavigationRail* method), 66

items (*kivymd.uix.dialog.MDDialog* attribute), 182

items (*kivymd.uix.menu.MDDropdownMenu* attribute), 39

items_spacing (*kivymd.uix.swiper.MDSwiper* attribute), 45

K

kivymd module, 322

kivymd.app module, 20

kivymd.color_definitions module, 22

kivymd.effects module, 324

kivymd.effects.roulettescroll module, 324

kivymd.effects.stiffscroll module, 326

kivymd.factory_registers module, 323

kivymd.font_definitions module, 27

kivymd.icon_definitions module, 25

kivymd.material_resources module, 323

kivymd.theming module, 6

kivymd.theming_dynamic_text module, 323

kivymd.toast module, 328

kivymd.toast.androidtoast module, 328

kivymd.toast.androidtoast.androidtoast module, 328

kivymd.toast.kivytoast module, 329

kivymd.toast.kivytoast.kivytoast module, 329

kivymd.tools module, 331

kivymd.tools.packaging module, 331

kivymd.tools.packaging.pyinstaller module, 331

kivymd.tools.packaging.pyinstaller.hook-kivymd module, 333

kivymd.tools.release module, 333

kivymd.tools.release.argument_parser module, 333

kivymd.tools.release.git_commands module, 334

kivymd.tools.release.make_release module, 334

kivymd.tools.release.update_icons module, 335

kivymd.uix module, 336

kivymd.uix.backdrop module, 77

kivymd.uix.banner module, 274

kivymd.uix.behaviors module, 337

kivymd.uix.behaviors.backgroundcolor_behavior module, 311

kivymd.uix.behaviors.elevation module, 286

kivymd.uix.behaviors.focus_behavior module, 300

kivymd.uix.behaviors.hover_behavior module, 284

kivymd.uix.behaviors.magic_behavior module, 307

kivymd.uix.behaviors.ripple_behavior module, 302

kivymd.uix.behaviors.toggle_behavior module, 305

kivymd.uix.behaviors.touch_behavior module, 309

kivymd.uix.bottomnavigation module, 268

kivymd.uix.bottomsheet module, 211

kivymd.uix.boxlayout
 module, 237

kivymd.uix.button
 module, 219

kivymd.uix.card
 module, 197

kivymd.uix.carousel
 module, 265

kivymd.uix.chip
 module, 81

kivymd.uix.circularlayout
 module, 127

kivymd.uix.datatables
 module, 47

kivymd.uix.dialog
 module, 179

kivymd.uix.dropdownitem
 module, 89

kivymd.uix.expansionpanel
 module, 261

kivymd.uix.filemanager
 module, 122

kivymd.uix.floatlayout
 module, 91

kivymd.uix.gridlayout
 module, 85

kivymd.uix.imagelist
 module, 189

kivymd.uix.label
 module, 161

kivymd.uix.list
 module, 92

kivymd.uix.menu
 module, 29

kivymd.uix.navigationdrawer
 module, 131

kivymd.uix.navigationrail
 module, 59

kivymd.uix.picker
 module, 140

kivymd.uix.progressbar
 module, 118

kivymd.uix.refreshlayout
 module, 129

kivymd.uix.relativelayout
 module, 237

kivymd.uix.screen
 module, 28

kivymd.uix.selection
 module, 278

kivymd.uix.selectioncontrol
 module, 192

kivymd.uix.slider
 module, 245

kivymd.uix.snackbar
 module, 239

kivymd.uix.spinner
 module, 234

kivymd.uix.stacklayout
 module, 267

kivymd.uix.swiper
 module, 41

kivymd.uix.tab
 module, 104

kivymd.uix.taptargetview
 module, 166

kivymd.uix.textfield
 module, 247

kivymd.uix.toolbar
 module, 67

kivymd.uix.tooltip
 module, 86

kivymd.utils
 module, 337

kivymd.utils.asynckivy
 module, 337

kivymd.utils.fitimage
 module, 337

kivymd.utils.fpsmonitor
 module, 341

kivymd.utils.hot_reload_viewer
 module, 341

kivymd_path (in *kivymd.tools.release.update_icons*), 335

L

label_check_texture_size()
(kivymd.toast.kivytoast.KivyToast.Toast method), 331

label_text_color (*kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute*), 231

lay_canvas_instructions()
(kivymd.uix.behaviors.ripple_behavior.CircularRippleBehavior method), 305

lay_canvas_instructions()
(kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 304

lay_canvas_instructions()
(kivymd.uix.behaviors.ripple_behavior.RectangularRippleBehavior method), 305

lay_canvas_instructions()
(kivymd.uix.button.MDRoundFlatButton method), 228

left_action (*kivymd.uix.bannerMDBanner attribute*), 277

left_action_items (*kivymd.uix.backdrop.MDBackdrop attribute*), 79

left_action_items (*kivymd.uix.toolbar.MDToolbar* attribute), 75
light_colors (*in module kivymd.color_definitions*), 24
line_anim (*kivymd.uix.textfield.MDTextField* attribute), 258
line_anim (*kivymd.uix.textfield.MDTextFieldRect* attribute), 256
line_color (*kivymd.uix.behaviors.backgroundcolor_behavior*.*BackgroundColorBehavior* attribute), 312
line_color (*kivymd.uix.button.MDRoundFlatButton* attribute), 228
line_color (*kivymd.uix.textfield.MDTextFieldRound* attribute), 260
line_color_focus (*kivymd.uix.textfield.MDTextField* attribute), 257
line_color_normal (*kivymd.uix.textfield.MDTextField* attribute), 257
line_width (*kivymd.uix.button.MDRoundFlatButton* attribute), 228
line_width (*kivymd.uix.spinner.MDSpinner* attribute), 236
lines (*kivymd.uix.imagelist.SmartTile* attribute), 191
lock_swiping (*kivymd.uix.tab.MDTabs* attribute), 116

M

magic_speed (*kivymd.uix.behaviors.magic_behavior*.*MagicBehavior* attribute), 309
MagicBehavior (class in *kivymd.uix.behaviors.magic_behavior*), 309
main() (*in module kivymd.tools.release.make_release*), 335
main() (*in module kivymd.tools.release.update_icons*), 336
make_icon_definitions() (*in module kivymd.tools.release.update_icons*), 336
max (*kivymd.effects.roulettescroll.RouletteScrollEffect* attribute), 325
max (*kivymd.effects.stiffscroll.StiffScrollEffect* attribute), 327
max_date (*kivymd.uix.picker.MDDatePicker* attribute), 158
max_degree (*kivymd.uix.circularlayout.MDCircularLayout* attribute), 128
max_friction (*kivymd.effects.stiffscroll.StiffScrollEffect* attribute), 327
max_height (*kivymd.uix.menu.MDDropdownMenu* attribute), 39
max_height (*kivymd.uix.textfield.MDTextField* attribute), 258
MAX_NAV_DRAWER_WIDTH (*in module kivymd.material_resources*), 323
max_opened_x (*kivymd.uix.card.MDCardSwipe* attribute), 209
max_swipe_x (*kivymd.uix.card.MDCardSwipe* attribute), 209
max_text_length (*kivymd.uix.textfield.MDTextField* attribute), 257
max_year (*kivymd.uix.picker.MDDatePicker* attribute), 158
md_bg_color (*kivymd.uix.behaviors.backgroundcolor_behavior*.*BackgroundColorBehavior* attribute), 312
md_bg_color (*kivymd.uix.button.MDFlatButton* attribute), 228
md_bg_color (*kivymd.uix.dialog.MDDialog* attribute), 188
md_bg_color (*kivymd.uix.toolbar.MDBottomAppBar* attribute), 76
md_icons (*in module kivymd.icon_definitions*), 27
MDActionBottomAppBarButton (class in *kivymd.uix.toolbar*), 74
MDActionTopAppBarButton (class in *kivymd.uix.toolbar*), 74
MDAdaptiveWidget (class in *kivymd.uix*), 336
MDApp (class in *kivymd.app*), 21
MDBackdrop (class in *kivymd.uix.backdrop*), 79
MDBackdropBackLayer (class in *kivymd.uix.backdrop*), 81
MDBackdropFrontLayer (class in *kivymd.uix.backdrop*), 81
MDBackdropToolbar (class in *kivymd.uix.backdrop*), 81
MDBanner (class in *kivymd.uix.banner*), 276
MDBottomAppBar (class in *kivymd.uix.toolbar*), 76
MDBottomNavigation (class in *kivymd.uix.bottomnavigation*), 272
MDBottomNavigationItem (class in *kivymd.uix.bottomnavigation*), 272
MDBottomSheet (class in *kivymd.uix.bottomsheet*), 216
MBoxLayout (class in *kivymd.uix.boxlayout*), 239
MCard (class in *kivymd.uix.card*), 207
MCardSwipe (class in *kivymd.uix.card*), 208
MCardSwipeFrontBox (class in *kivymd.uix.card*), 210
MCardSwipeLayerBox (class in *kivymd.uix.card*), 210
MCarousel (class in *kivymd.uix.carousel*), 266
MDCheckbox (class in *kivymd.uix.selectioncontrol*), 196
MDChip (class in *kivymd.uix.chip*), 83
MDChooseChip (class in *kivymd.uix.chip*), 84
MDCircularLayout (class in *kivymd.uix.circularlayout*), 128
MDCustomBottomSheet (class in *kivymd.uix.bottomsheet*), 217
MDDataTable (class in *kivymd.uix.datatables*), 48
MDDatePicker (class in *kivymd.uix.picker*), 157
MDDialog (class in *kivymd.uix.dialog*), 180
MDDropDownItem (class in *kivymd.uix.dropdownitem*), 90
MDDropdownMenu (class in *kivymd.uix.menu*), 39
MDExpansionPanel (class in *kivymd.uix.expansionpanel*), 90

| | | |
|--|-----------------|--|
| <code>kivymd.uix.expansionpanel), 263</code> | | |
| <code>MDEExpansionPanelLabel (class
kivymd.uix.expansionpanel), 263</code> | <code>in</code> | <code>MDSwiper (class in kivymd.uix.swiper), 44</code> |
| <code>MDEExpansionPanelOneLine (class
kivymd.uix.expansionpanel), 263</code> | <code>in</code> | <code>MDSwiperItem (class in kivymd.uix.swiper), 44</code> |
| <code>MDEExpansionPanelThreeLine (class
kivymd.uix.expansionpanel), 263</code> | <code>in</code> | <code>MDSwitch (class in kivymd.uix.selectioncontrol), 196</code> |
| <code>MDEExpansionPanelTwoLine (class
kivymd.uix.expansionpanel), 263</code> | <code>in</code> | <code>MDTab (class in kivymd.uix.bottomnavigation), 272</code> |
| <code>MDFFileManager (class in kivymd.uix.filemanager), 125</code> | | <code>MDTabs (class in kivymd.uix.tab), 115</code> |
| <code>MDFillRoundFlatButton (class in kivymd.uix.button),
229</code> | | <code>MDTabsBase (class in kivymd.uix.tab), 114</code> |
| <code>MDFillRoundFlatIconButton (class
kivymd.uix.button), 229</code> | <code>in</code> | <code>MDTapTargetView (class in kivymd.uix.taptargetview),
175</code> |
| <code>MDFloatButton (class in kivymd.uix.button), 228</code> | | <code>MDTextButton (class in kivymd.uix.button), 230</code> |
| <code>MDFloatingActionButton (class in kivymd.uix.button),
230</code> | | <code>MDTextField (class in kivymd.uix.textfield), 256</code> |
| <code>MDFloatingActionButtonSpeedDial (class
kivymd.uix.button), 231</code> | <code>in</code> | <code>MDTextFieldRect (class in kivymd.uix.textfield), 256</code> |
| <code>MDFloatLayout (class in kivymd.uix.floatlayout), 92</code> | | <code>MDTextFieldRound (class in kivymd.uix.textfield), 259</code> |
| <code>MDGridBottomSheet (class in kivymd.uix.bottomsheet),
219</code> | | <code>MDThemePicker (class in kivymd.uix.picker), 160</code> |
| <code>MDGridLayout (class in kivymd.uix.gridlayout), 86</code> | | <code>MDTimePicker (class in kivymd.uix.picker), 159</code> |
| <code>MDIcon (class in kivymd.uix.label), 165</code> | | <code>MDToggleButton (class
kivymd.uix.behaviors.toggle_behavior), 307</code> |
| <code>MDIconButton (class in kivymd.uix.button), 229</code> | | <code>MDToolbar (class in kivymd.uix.toolbar), 74</code> |
| <code>MDLabel (class in kivymd.uix.label), 165</code> | | <code>MDTooltip (class in kivymd.uix.tooltip), 88</code> |
| <code>MDList (class in kivymd.uix.list), 102</code> | | <code>MDTooltipViewClass (class in kivymd.uix.tooltip), 89</code> |
| <code>MDListBottomSheet (class in kivymd.uix.bottomsheet),
218</code> | | <code>min (kivymd.effects.roulettescroll.RouletteScrollEffect
attribute), 325</code> |
| <code>MDDNavigationDrawer (class
kivymd.uix.navigationdrawer), 137</code> | <code>in</code> | <code>min (kivymd.effects.stiffscroll.StiffScrollEffect attribute),
327</code> |
| <code>MDDNavigationLayout (class
kivymd.uix.navigationdrawer), 137</code> | <code>in</code> | <code>min_date (kivymd.uix.picker.MDDDatePicker attribute),
158</code> |
| <code>MDDNavigationRail (class in kivymd.uix.navigationrail),
62</code> | | <code>min_year (kivymd.uix.picker.MDDDatePicker attribute),
158</code> |
| <code>MDPProgressBar (class in kivymd.uix.progressbar), 121</code> | | <code>minute (kivymd.uix.picker.MDTimePicker attribute), 159</code> |
| <code>MDRaisedButton (class in kivymd.uix.button), 228</code> | | <code>minute_radius (kivymd.uix.picker.MDTimePicker
attribute), 159</code> |
| <code>MDRectangleFlatButton (class in kivymd.uix.button),
228</code> | | <code>mode (kivymd.uix.picker.MDDDatePicker attribute), 158</code> |
| <code>MDRectangleFlatButtonIconButton (class
kivymd.uix.button), 228</code> | <code>in</code> | <code>mode (kivymd.uix.textfield.MDTextField attribute), 257</code> |
| <code>MDRelativeLayout (class in kivymd.uix.relativelayout),
237</code> | | <code>mode (kivymd.uix.toolbar.MDToolbar attribute), 75</code> |
| <code>MDRoundFlatButton (class in kivymd.uix.button), 228</code> | | <code>module
kivymd, 322</code> |
| <code>MDRoundFlatButtonIconButton (class in kivymd.uix.button),
228</code> | | <code>kivymd.app, 20</code> |
| <code>MDScreen (class in kivymd.uix.screen), 29</code> | | <code>kivymd.color_definitions, 22</code> |
| <code>MDScrollViewRefreshLayout (class
kivymd.uix.refreshlayout), 131</code> | <code>in</code> | <code>kivymd.effects, 324</code> |
| <code>MDSSelectionList (class in kivymd.uix.selection), 283</code> | | <code>kivymd.effects.roulettescroll, 324</code> |
| <code>MDSeparator (class in kivymd.uix.card), 207</code> | | <code>kivymd.effects.stiffscroll, 326</code> |
| <code>MDSlider (class in kivymd.uix.slider), 246</code> | | <code>kivymd.factory_registers, 323</code> |
| <code>MDSpinner (class in kivymd.uix.spinner), 236</code> | | <code>kivymd.font_definitions, 27</code> |
| <code>MDStackLayout (class in kivymd.uix.stacklayout), 268</code> | | <code>kivymd.icon_definitions, 25</code> |
| | | <code>kivymd.material_resources, 323</code> |
| | | <code>kivymd.theming, 6</code> |
| | | <code>kivymd.theming_dynamic_text, 323</code> |
| | | <code>kivymd.toast, 328</code> |
| | | <code>kivymd.toast.androidtoast, 328</code> |
| | | <code>kivymd.toast.androidtoast.androidtoast,
328</code> |
| | | <code>kivymd.toast.kivytoast, 329</code> |
| | | <code>kivymd.toast.kivytoast.kivytoast, 329</code> |
| | | <code>kivymd.tools, 331</code> |
| | | <code>kivymd.tools.packaging, 331</code> |

kivymd.tools.packaging.pyinstaller, 331
kivymd.tools.packaging.pyinstaller.hook-kivymd, 333
kivymd.tools.release, 333
kivymd.tools.release.argument_parser, 333
kivymd.tools.release.git_commands, 334
kivymd.tools.release.make_release, 334
kivymd.tools.release.update_icons, 335
kivymd.uix, 336
kivymd.uix.backdrop, 77
kivymd.uix.banner, 274
kivymd.uix.behaviors, 337
kivymd.uix.behaviors.backgroundcolor_behavior, 311
kivymd.uix.behaviors.elevation, 286
kivymd.uix.behaviors.focus_behavior, 300
kivymd.uix.behaviors.hover_behavior, 284
kivymd.uix.behaviors.magic_behavior, 307
kivymd.uix.behaviors.ripple_behavior, 302
kivymd.uix.behaviors.toggle_behavior, 305
kivymd.uix.behaviors.touch_behavior, 309
kivymd.uix.bottomnavigation, 268
kivymd.uix.bottomsheet, 211
kivymd.uix.boxlayout, 237
kivymd.uix.button, 219
kivymd.uix.card, 197
kivymd.uix.carousel, 265
kivymd.uix.chip, 81
kivymd.uix.circularlayout, 127
kivymd.uix.datatables, 47
kivymd.uix.dialog, 179
kivymd.uix.dropdownitem, 89
kivymd.uix.expansionpanel, 261
kivymd.uix.filemanager, 122
kivymd.uix.floatlayout, 91
kivymd.uix.gridlayout, 85
kivymd.uix.imagelist, 189
kivymd.uix.label, 161
kivymd.uix.list, 92
kivymd.uix.menu, 29
kivymd.uix.navigationdrawer, 131
kivymd.uix.navigationrail, 59
kivymd.uix.picker, 140
kivymd.uix.progressbar, 118
kivymd.uix.refreshlayout, 129
kivymd.uix.relativelayout, 237
kivymd.uix.screen, 28
kivymd.uix.selection, 278
kivymd.uix.selectioncontrol, 192
kivymd.uix.slider, 245
kivymd.uix.snackbar, 239
kivymd.uix.spinner, 234
kivymd.uix.stacklayout, 267
kivymd.uix.swiper, 41
kivymd.uix.tab, 104
kivymd.uix.taptargetview, 166
kivymd.uix.textfield, 247
kivymd.uix.toolbar, 67
kivymd.uix.tooltip, 86
kivymd.utils, 337
kivymd.utils.asynckivy, 337
kivymd.utils.fitimage, 337
kivymd.utils.fpsmonitor, 341
kivymd.utils.hot_reload_viewer, 341
month (kivymd.uix_picker.MDDatePicker attribute), 158
move_changelog() (in module kivymd.tools.release.make_release), 335

N

near_next_notch() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326
near_notch() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326
nearest_notch() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326
next_notch() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326
no_ripple_effect (kivymd.uix.tab.MDTabs attribute), 116
normal_color (kivymd.uix.textfield.MDTextFieldRound attribute), 260
notch_center_x (kivymd.uix.toolbar.NotchedBox attribute), 74
notch_radius (kivymd.uix.toolbar.NotchedBox attribute), 74
NotchedBox (class in kivymd.uix.toolbar), 74

O

ObservableShadow (class in kivymd.uix.behaviors.elevation), 299
on_hint_text() (kivymd.uix.textfield.MDTextField method), 259
on_is_off() (kivymd.uix.slider.MDSlider method), 247
on_rotation_angle() (kivymd.uix.spinner.MDSpinner method), 236
on_shadow_pos() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 298
on_action_button() (kivymd.uix.navigationrail.MDNavigationRail method), 66
on_action_button() (kivymd.uix.toolbar.MDToolbar method), 76
on_active() (kivymd.uix.selectioncontrol.MDCheckbox method), 196
on_active() (kivymd.uix.slider.MDSlider method), 247
on_active() (kivymd.uix.spinner.MDSpinner method), 236

on_adaptive_height() (*kivymd.uix.MDAdaptiveWidget method*), 336

on_adaptive_size() (*kivymd.uix.MDAdaptiveWidget method*), 336

on_adaptive_width() (*kivymd.uix.MDAdaptiveWidget method*), 336

on_anchor() (*kivymd.uix.card.MDCardSwipe method*), 209

on_any_event() (*kivymd.utils.hot_reload_viewer.HotReloadHandler kivymd.uix.picker.MDDatePicker method*), 343

on_bg_color_root_button() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_bg_color_stack_button() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_bg_hint_color() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_cancel() (*kivymd.uix.picker.BaseDialogPicker method*), 157

on_carousel_index() (*kivymd.uix.tab.MDTabs method*), 118

on_check_press() (*kivymd.uix.datatables.MDDDataTable method*), 59

on_close() (*kivymd.uix.backdrop.MDBackdrop method*), 80

on_close() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_close() (*kivymd.uix.expansionpanel.MDExpansionPanel method*), 264

on_close() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66

on_close() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178

on_coasted_to_stop() (*kivymd.effects.roulettescroll.RouletteScrollEffect method*), 326

on_color_active() (*kivymd.uix.textfield.MDTextFieldRound method*), 261

on_color_icon_root_button() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_color_icon_stack_button() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_color_mode() (*kivymd.uix.textfield.MDTextField method*), 259

on_data() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233

on_description_text() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178

on_description_text_bold() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178

on_determinate_complete() (*kivymd.uix.spinner.MDSpinner method*), 236

on_device_orientation() (*kivymd.uix.behaviors.elevation.CommonElevationBehavior method*), 158

on_disabled() (*kivymd.uix.button.MDTextButton method*), 231

on_disabled() (*kivymd.uix.button.MDFillRoundFlatIconButton method*), 229

on_disabled() (*kivymd.uix.textfield.MDTextField method*), 259

on_dismiss() (*kivymd.uix.menu.MDDropdownMenu method*), 40

on_double_tap() (*kivymd.uix.behaviors.touch_behavior.TouchBehavior method*), 310

on_draw_shadow() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178

on_elevation() (*kivymd.uix.behaviors.elevation.CommonElevationBehavior method*), 298

on_error() (*kivymd.utils.hot_reload_viewer.HotReloadViewer method*), 343

on_enter() (*kivymd.uix.tooltip.MDTooltip method*), 89

on_focus() (*kivymd.uix.textfield.MDTextField method*), 259

on_focus() (*kivymd.uix.textfield.MDTextFieldRound method*), 260

on_font_name_helper_text() (*kivymd.uix.textfield.MDTextField method*), 259

on_font_name_hint_text() (*kivymd.uix.textfield.MDTextField method*), 259

on_font_name_max_length() (*kivymd.uix.textfield.MDTextField method*), 259

on_header() (*kivymd.uix.backdrop.MDBackdrop method*), 80

on_header_cls() (*kivymd.uix.menu.MDDropdownMenu method*), 40
on_height() (*kivymd.uix.textfield.MDTextField method*), 259
on_hint() (*kivymd.uix.slider.MDSlider method*), 247
on_hint_animation() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233
on_hint_text() (*kivymd.uix.textfield.MDTextField method*), 259
on_icon() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233
on_icon() (*kivymd.uix.chip.MDChip method*), 84
on_icon() (*kivymd.uix.toolbar.MDToolbar method*), 76
on_icon_color() (*kivymd.uix.button.MDRoundFlatButtonIcon method*), 229
on_icon_color() (*kivymd.uix.toolbar.MDToolbar method*), 76
on_icon_left() (*kivymd.uix.textfield.MDTextFieldRound method*), 260
on_icon_left_color() (*kivymd.uix.textfield.MDTextFieldRound method*), 260
on_icon_right() (*kivymd.uix.textfield.MDTextFieldRound method*), 259
on_icon_right() (*kivymd.uix.textfield.MDTextFieldRound method*), 260
on_icon_right_color() (*kivymd.uix.textfield.MDTextFieldRound method*), 259
on_item_switch() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
on_label_text_color() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233
on_leave() (*kivymd.uix.behaviors.focus_behavior.FocusBehavior method*), 302
on_leave() (*kivymd.uix.behaviors.hover_behavior.HoverBehavior method*), 286
on_leave() (*kivymd.uix.bottomnavigation.MDBottomNavigation method*), 272
on_leave() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233
on_leave() (*kivymd.uix.tooltip.MDTooltip method*), 89
on_left_action_items() (*kivymd.uix.backdrop.MDBackdrop method*), 80
on_left_action_items() (*kivymd.uix.toolbar.MDToolbar method*), 76
on_line_color_focus()
on_long_touch() (*kivymd.uix.behaviors.touch_behavior.TouchBehavior method*), 310
on_long_touch() (*kivymd.uix.tooltip.MDTooltip method*), 89
on_md_bg_color() (*kivymd.uix.button.MDFillRoundFlatButton method*), 229
on_md_bg_color() (*kivymd.uix.button.MDFillRoundFlatIconButton method*), 229
on_md_bg_color() (*kivymd.uix.button.MDTextButton method*), 231
on_md_bg_color() (*kivymd.uix.toolbar.MDToolbar method*), 76
on_md_bg_color() (*kivymd.uix.toolbar.MDToolbar method*), 76
on_mouse_update() (*kivymd.uix.behaviors.hover_behavior.HoverBehavior method*), 286
on_notch() (*kivymd.effects.roulettescroll.RouletteScrollEffect method*), 326
on_open() (*kivymd.toast.kivytoast.kivytoast.Toast method*), 331
on_open() (*kivymd.uix.backdrop.MDBackdrop method*), 80
on_open() (*kivymd.uix.button.MDFloatingActionButtonSpeedDial method*), 233
on_open() (*kivymd.uix.dialog.MDDialog method*), 188
on_open() (*kivymd.uix.expansionpanel.MDExpansionPanel method*), 264
on_open() (*kivymd.uix.navigationrail.MDNavigationRail method*), 66
on_open() (*kivymd.uix.picker.MDThemePicker method*), 160
on_open() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178
on_open_progress() (*kivymd.uix.card.MDCardSwipe method*), 209
on_outer_touch() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178
on_outer_touch() (*kivymd.uix.taptargetview.MDTapTargetView method*), 178
on_outside_click() (*kivymd.uix.taptargetview.MDTapTargetView method*), 179
on_overswipe_left() (*kivymd.uix.swiper.MDSwiper method*), 46
on_overswipe_right() (*kivymd.uix.swiper.MDSwiper method*), 46
on_palette() (*kivymd.uix.spinner.MDSpinner method*), 236
on_panel_color() (*kivymd.uix.bottomnavigation.MDBottomNavigation method*), 273

on_path() (*kivymd.utils.hot_reload_viewer.HotReloadView* method), 343

on_pre_swipe() (*kivymd.uix.swiper.MDSwiper* method), 46

on_press() (*kivymd.uix.button.MDTextButton* method), 231

on_radius() (*kivymd.uix.navigationdrawer.MDNavigationDrawer* method), 140

on_rail_state() (*kivymd.uix.navigationrail.MDNavigationRail* method), 66

on_ref_press() (*kivymd.uix.tab.MDTabs* method), 118

on_resize() (*kivymd.uix.bottomnavigation.MDBottomNavigation* method), 273

on_right_action_items() (*kivymd.uix.toolbar.MDToolbar* method), 76

on_ripple_behavior() (*kivymd.uix.card.MDCard* method), 208

on_row_press() (*kivymd.uix.datatables.MDDDataTable* method), 59

on_save() (*kivymd.uix_picker.BaseDialogPicker* method), 157

on_scroll_start() (*kivymd.uix.swiper.MDSwiper* method), 46

on_selected() (*kivymd.uix.selection.MDSelectionList* method), 284

on_shadow_group() (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* method), 298

on_shadow_pos() (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* method), 299

on_show() (*kivymd.uix.tooltip.MDTooltip* method), 89

on_show_off() (*kivymd.uix.slider.MDSlider* method), 247

on_size() (*kivymd.uix.bottomnavigation.MDBottomNavigation* method), 273

on_size() (*kivymd.uix.selectioncontrol.MDSwitch* method), 197

on_size() (*kivymd.uix.tab.MDTabs* method), 118

on_slide_complete() (*kivymd.uix.carousel.MDCarousel* method), 266

on_slide_progress() (*kivymd.uix.carousel.MDCarousel* method), 266

on_slide_progress() (*kivymd.uix.tab.MDTabs* method), 118

on_stars() (*kivymd.uix.imagelist.SmartTileWithStar* method), 192

on_state() (*kivymd.uix.selectioncontrol.MDCheckbox* method), 196

on_swipe() (*kivymd.uix.swiper.MDSwiper* method), 46

on_swipe_complete() (*kivymd.uix.card.MDCardSwipe* method), 209

on_swipe_left() (*kivymd.uix.swiper.MDSwiper* method), 46

on_swipe_right() (*kivymd.uix.swiper.MDSwiper* method), 46

on_tab_press() (*kivymd.uix.bottomnavigation.MDBottomNavigationItem* method), 272

on_tab_press() (*kivymd.uix.bottomnavigation.MDTab* method), 272

on_tab_release() (*kivymd.uix.bottomnavigation.MDTab* method), 272

on_tab_switch() (*kivymd.uix_picker.MDThemePicker* method), 160

on_tab_switch() (*kivymd.uix.tab.MDTabs* method), 118

on_tab_touch_down() (*kivymd.uix.bottomnavigation.MDTab* method), 272

on_tab_touch_move() (*kivymd.uix.bottomnavigation.MDTab* method), 272

on_tab_touch_up() (*kivymd.uix.bottomnavigation.MDTab* method), 272

on_target_radius() (*kivymd.uix_taptargetview.MDTapTargetView* method), 178

on_target_touch() (*kivymd.uix_taptargetview.MDTapTargetView* method), 178

on_text_color() (*kivymd.uix_label.MDLabel* method), 165

on_text_color_active() (*kivymd.uix_bottomnavigation.MDBottomNavigation* method), 273

on_text_color_normal() (*kivymd.uix_bottomnavigation.MDBottomNavigation* method), 273

on_text_validate() (*kivymd.uix_textfield.MDTextField* method), 259

on_theme_style() (*kivymd_theming.ThemeManager* method), 18

on_theme_text_color() (*kivymd.uix_label.MDLabel* method), 165

on_title_text() (*kivymd.uix_taptargetview.MDTapTargetView* method), 178

on_title_text_bold() (*kivymd.uix_taptargetview.MDTapTargetView* method), 178

on_title_text_size() (*kivymd.uix_taptargetview.MDTapTargetView* method), 178

on_touch_down() (*kivymd.toast.kivytoast.Toast* method), 331

on_touch_down() (*kivymd.uix_behaviors.ripple_behavior.CommonRipple*

method), 305
on_touch_down() (kivymd.uix.button.MDFloatingActionButton method), 230
on_touch_down() (kivymd.uix.card.MDCardSwipe method), 210
on_touch_down() (kivymd.uix.carousel.MDCarousel method), 266
on_touch_down() (kivymd.uix.chip.MDChip method), 84
on_touch_down() (kivymd.uix.menu.MDDropdownMenu method), 40
on_touch_down() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 139
on_touch_down() (kivymd.uix.slider.MDSlider method), 247
on_touch_down() (kivymd.uix.swiper.MDSwiper method), 46
on_touch_move() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 305
on_touch_move() (kivymd.uix.button.MDFloatingActionButton method), 230
on_touch_move() (kivymd.uix.card.MDCardSwipe method), 209
on_touch_move() (kivymd.uix.menu.MDDropdownMenu method), 40
on_touch_move() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 139
on_touch_up() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 309
on_touch_up() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 305
on_touch_up() (kivymd.uix.button.MDFloatingActionButton method), 230
on_touch_up() (kivymd.uix.card.MDCardSwipe method), 210
on_touch_up() (kivymd.uix.carousel.MDCarousel method), 266
on_touch_up() (kivymd.uix.menu.MDDropdownMenu method), 40
on_touch_up() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 140
on_touch_up() (kivymd.uix.refreshlayout.MDScrollViewRefreshLayout method), 131
on_touch_up() (kivymd.uix.slider.MDSlider method), 247
on_touch_up() (kivymd.uix.swiper.MDSwiper method), 47
on_triple_tap() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 310
on_type() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 140
on_type() (kivymd.uix.toolbar.MDToolbar method), 75
on_unselected() (kivymd.uix.selection.MDSelectionList method), 284
on_useActionButton() (kivymd.uix.navigationrail.MDNavigationRail method), 66
on_useResizable() (kivymd.uix.navigationrail.MDNavigationRail method), 66
on_useTitle() (kivymd.uix.navigationrail.MDNavigationRail method), 66
on_value() (kivymd.effects.stiffscroll.StiffScrollEffect method), 327
on_value_normalized() (kivymd.uix.slider.MDSlider method), 247
onVisible() (kivymd.uix.navigationrail.MDNavigationRail method), 66
onWidth() (kivymd.uix.textfield.MDTextField method), 259
OneLineAvatarIconListItem (class in kivymd.uix.list), 103
OneLineAvatarListItem (class in kivymd.uix.list), 103
OneLineIconListItem (class in kivymd.uix.list), 103
OneLineListItemIcon (class in kivymd.uix.list), 103
open() (kivymd.uix.backdrop.MDBackdrop method), 80
open() (kivymd.uix.bottomsheet.MDBottomSheet method), 217
open() (kivymd.uix.menu.MDDropdownMenu method), 40
open() (kivymd.uix.navigationrail.MDNavigationRail method), 66
open_card() (kivymd.uix.card.MDCardSwipe method), 210
open_panel() (kivymd.uix.expansionpanel.MDExpansionPanel method), 264
open_progress (kivymd.uix.card.MDCardSwipe attribute), 208
open_progress (kivymd.uix.navigationdrawer.MDNavigationDrawer attribute), 138
open_stack() (kivymd.uix.button.MDFloatingActionButtonSpeedDial method), 233
opening_time (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 232
opening_time (kivymd.uix.card.MDCardSwipe attribute), 208
opening_time (kivymd.uix.expansionpanel.MDExpansionPanel attribute), 264
opening_time (kivymd.uix.menu.MDDropdownMenu attribute), 40
opening_time (kivymd.uix.navigationdrawer.MDNavigationDrawer attribute), 139
opening_time_button_rotation (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 232
opening_transition (kivymd.uix.banner.MDBanner

attribute), 277
opening_transition(kivymd.uix.button.MDFloatingActionSpeedDial_height
attribute), 232
opening_transition (kivymd.uix.card.MDCardSwipe_trIBUTE), 57
attribute), 208
opening_transition(kivymd.uix.expansionpanel.MDExpansionPanel_pagination_menu_pos
attribute), 264
opening_transition(kivymd.uix.menu.MDDropdownMenu_palette (in module kivymd.color_definitions), 24
attribute), 40
opening_transition(kivymd.uix.navigationdrawer.MDNavigateDrawer_pagination_attributes(kivymd.uix.expansionpanel.MDExpansionPanel
attribute), 139
opening_transition_button_rotation panel_color (kivymd.uix.bottomnavigation.TabbedPanelBase
(kivymd.button.MDFloatingActionButtonSpeedDial attribute), 272
attribute), 232
opposite_bg_dark parent_background (kivymd.uix.label.MDLabel
attribute), 15
opposite_bg_darkest parse_args() (kivymd.tools.release.argument_parser.ArgumentParserWith
(kivymd.theming.ThemeManager attribute), 15
opposite_bg_light path (in module kivymd), 322
attribute), 16
opposite_bg_normal position (kivymd.uix.menu.MDDropdownMenu at-
attribute), 15
opposite_colors press_floating_action_button()
attribute), 20
opposite_colors (kivymd.uix.button.MDFillRoundFlatButton method), 66
attribute), 229
opposite_colors (kivymd.uix.toolbar.MDToolbar attribute), 75
opposite_disabled_hint_text_color preview (kivymd.uix.filemanager.MDFileManager
(kivymd.theming.ThemeManager attribute), 17
opposite_divider_color attribute), 126
opposite_icon_color primary_color (kivymd.theming.ThemeManager
(kivymd.theming.ThemeManager attribute), 16
opposite_secondary_text_color attribute), 11
opposite_text_color primary_color (kivymd.uix.picker.BaseDialogPicker
(kivymd.theming.ThemeManager attribute), 16
orientation primary_dark (kivymd.theming.ThemeManager at-
attribute), 121
outer_circle_alpha primary_dark_hue (kivymd.theming.ThemeManager
attribute), 176
outer_circle_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 11
outer_radius primary_hue (kivymd.theming.ThemeManager at-
attribute), 175
over_widget primary_light (kivymd.theming.ThemeManager
(kivymd.uix.bannerMDBanner attribute), 277
overlap primary_light_hue (kivymd.theming.ThemeManager
(kivymd.uix.imagelist.SmartTile attribute), 191
overlay_color primary_palette (kivymd.theming.ThemeManager at-
attribute), 283
P
padding (kivymd.uix.backdrop.MDBackdrop attribute), 79
padding_inset (kivymd.uix.datatables.MDDatatable attribute), 56
parent_inset (kivymd.uix.datatables.MDDatatable attribute), 57
pagination_menu_pos (kivymd.uix.datatables.MDDatatable attribute), 56
panel_color (kivymd.uix.bottomnavigation.TabbedPanelBase attribute), 272
parent_background (kivymd.uix.label.MDLabel attribute), 165
parse_args() (kivymd.tools.release.argument_parser.ArgumentParserWith method), 334
path (in module kivymd), 322
path (kivymd.utils.hot_reload_viewer.HotReloadViewer attribute), 343
position (kivymd.uix.menu.MDDropdownMenu attribute), 40
press_floating_action_button() (kivymd.uix.navigationrail.MDNavigationRail attribute), 66
preview (kivymd.uix.filemanager.MDFileManager attribute), 126
previous_tab (kivymd.uix.bottomnavigation.TabbedPanelBase attribute), 272
primary_color (kivymd.theming.ThemeManager attribute), 11
primary_color (kivymd.uix.picker.BaseDialogPicker attribute), 147
primary_dark (kivymd.theming.ThemeManager attribute), 12
primary_dark_hue (kivymd.theming.ThemeManager attribute), 11
primary_hue (kivymd.theming.ThemeManager attribute), 10
primary_light (kivymd.theming.ThemeManager attribute), 11
primary_light_hue (kivymd.theming.ThemeManager attribute), 9
primary_palette (kivymd.theming.ThemeManager attribute), 9
progress_round_color (kivymd.uix.selection.MDSelectionList attribute), 283
progress_round_size (kivymd.uix.selection.MDSelectionList attribute), 283
pull_back_velocity (kivymd.effects.roulettescroll.RouletteScrollEffect attribute), 325
pull_duration (kivymd.effects.roulettescroll.RouletteScrollEffect attribute), 325

attribute), 325

R

r (in module kivymd.factory_registers), 323

r (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundAttribute), 311

radio_icon_down (kivymd.uix.selectioncontrol.MDCheckbox attribute), 196

radio_icon_normal (kivymd.uix.selectioncontrol.MDCheckbox attribute), 196

radius (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundAttribute), 311

radius (kivymd.uix.behaviors.elevation.CommonElevationAttribute), 295

radius (kivymd.uix.bottomsheetMDBottomSheet attribute), 216

radius (kivymd.uix.chip.MDChip attribute), 84

radius (kivymd.uix.menu.MDDropdownMenu attribute), 40

radius (kivymd.uix.picker.BaseDialogPicker attribute), 147

radius (kivymd.uix.textfield.MDTextField attribute), 258

radius (kivymd.utils.fitimage.FitImage attribute), 340

radius_from (kivymd.uix.bottomsheetMDBottomSheet attribute), 216

radius_left (kivymd.uix.backdrop.MDBackdrop attribute), 80

radius_right (kivymd.uix.backdrop.MDBackdrop attribute), 80

rail_state (kivymd.uix.navigationrail.MDNavigationRail attribute), 66

re_additional_icons (in module kivymd.tools.release.update_icons), 335

re_icon_definitions (in module kivymd.tools.release.update_icons), 336

re_icons_json (in module kivymd.tools.release.update_icons), 335

re_quote_keys (in module kivymd.tools.release.update_icons), 336

re_version (in module kivymd.tools.release.update_icons), 336

re_version_in_file (in module kivymd.tools.release.update_icons), 336

RectangularElevationBehavior (class in kivymd.uix.behaviors.elevation), 299

RectangularRippleBehavior (class in kivymd.uix.behaviors.ripple_behavior), 305

refresh_done() (kivymd.uix.refreshlayout.MDScrollViewRefreshLayout method), 131

refresh_tabs() (kivymd.uix.bottomnavigation.MDBottomNavigation method), 273

RefreshSpinner (class in kivymd.uix.refreshlayout), 131

release (in module kivymd), 322

reload() (kivymd.uix.imagelist.SmartTile method), 192

reload() (kivymd.utils.fitimage.FitImage method), 340

remove_label() (kivymd.uix.button.MDRectangleFlatIconButton method), 228

remove_label() (kivymd.uix.button.MDRoundFlatButton method), 229

remove_notch() (kivymd.uix.toolbar.MDToolbar method), 76

remove_shadow() (kivymd.uix.toolbar.MDToolbar method), 76

remove_tooltip() (kivymd.uix.tooltip.MDTooltip method), 89

remove_widget() (kivymd.uix.bottomnavigation.MDBottomNavigation method), 273

remove_widget() (kivymd.uix.circularlayout.MDCircularLayout method), 128

remove_widget() (kivymd.uix.list.MDList method), 102

remove_widget() (kivymd.uix.swiper.MDSwiper method), 46

remove_widget() (kivymd.uix.tab.MDTabs method), 117

replace_in_file() (in module kivymd.tools.release.make_release), 335

required (kivymd.uix.textfield.MDTextField attribute), 257

resize_content_layout() (kivymd.uix.bottomsheet.MDBottomSheet method), 217

reversed (kivymd.uix.progressbar.MDProgressBar attribute), 121

right_action (kivymd.uix.banner.MDBanner attribute), 277

right_action_items (kivymd.uix.backdrop.MDBackdrop attribute), 79

right_action_items (kivymd.uix.toolbar.MDToolbar attribute), 75

right_pad (kivymd.uix.button.MDFloatingActionButtonSpeedDial attribute), 231

ripple_alpha (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_behavior (kivymd.uix.card.MDCard attribute), 208

ripple_color (kivymd.theming.ThemeManager attribute), 17

ripple_color (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_duration (kivymd.uix.tab.MDTabs attribute), 116

ripple_duration_in_fast

ripple_duration_in_slow (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_duration_out
 (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_func_in (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_func_out (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_rad_default (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 304

ripple_scale (kivymd.uix.behaviors.ripple_behavior.Circular Ripple Behavior attribute), 305

ripple_scale (kivymd.uix.behaviors.ripple_behavior.Rectangular Ripple Behavior attribute), 305

root_button_anim (kivymd.uix.button.MDFloatingActionButtonSpeedDial), 284

root_layout (kivymd.uix.refreshlayout.MDScrollViewRefreshLayout), 84

RouletteScrollEffect (class in kivymd.effects.roulettescroll), 325

round (kivymd.uix.toolbar.MDToolbar attribute), 75

RoundedRectangularElevationBehavior (class in kivymd.uix.behaviors.elevation), 299

row_data (kivymd.uix.datatables.MDDDataTable attribute), 53

row_spacing (kivymd.uix.circularlayout.MDCircularLayout attribute), 128

rows_num (kivymd.uix.datatables.MDDDataTable attribute), 56

run_pre_commit() (in module kivymd.tools.release.make_release), 335

running_away() (kivymd.uix.progressbar.MDProgressBar method), 122

running_duration (kivymd.uix.progressbar.MDProgressBar attribute), 122

running_transition (kivymd.uix.progressbar.MDProgressBar attribute), 122

S

screen (kivymd.uix.bottomsheet.MDCustomBottomSheet attribute), 218

scrim_alpha_transition (kivymd.uix.navigationdrawer.MDNavigationDrawer attribute), 139

scrim_color (kivymd.uix.navigationdrawer.MDNavigationDrawer attribute), 139

scroll (kivymd.effects.stiffscroll.StiffScrollEffect attribute), 327

search (kivymd.uix.filemanager.MDFileManager attribute), 126

secondary_text_color (kivymd.theming.ThemeManager attribute), 16

sel_day (kivymd.uix.picker.MDDatePicker attribute), 158

sel_month (kivymd.uix.picker.MDDatePicker attribute), 158

sel_year (kivymd.uix.picker.MDDatePicker attribute), 158

select_dir_or_file()
 (kivymd.uix.filemanager.MDFileManager method), 127

select_file_behavior_on_press_button()
 (kivymd.uix.filemanager.MDFileManager method), 125

selected_all() (kivymd.uix.selection.MDSelectionList method), 125

selected_chip_color (kivymd.uix.chip.MDChip attribute), 196

selected_mode (kivymd.uix.selection.MDSelectionList attribute), 283

selection (kivymd.uix.filemanager.MDFileManager attribute), 126

selector (kivymd.uix.filemanager.MDFileManager attribute), 126

set_action_color_button()
 (kivymd.uix.navigationrail.MDNavigationRail method), 66

set_action_icon_button()
 (kivymd.uix.navigationrail.MDNavigationRail method), 67

set_action_text_button()
 (kivymd.uix.navigationrail.MDNavigationRail method), 67

set_box_title_size()
 (kivymd.uix.navigationrail.MDNavigationRail method), 67

set_chevron_down() (kivymd.uix.expansionpanel.MDExpansionPanel method), 264

set_chevron_up() (kivymd.uix.expansionpanel.MDExpansionPanel method), 264

set_clearcolor (kivymd.theming.ThemeManager attribute), 17

set_clearcolor_by_theme_style()
 (kivymd.theming.ThemeManager method), 18

set_color() (kivymd.uix.chip.MDChip method), 84

set_color_menu_item()
 (kivymd.uix.navigationrail.MDNavigationRail method), 67

set_colors() (kivymd.theming.ThemeManager method), 18

method), 18
set_current() (kivymd.uix.swiper.MDSwiper method), 46
set_icon_color() (kivymd.uix.button.MDFillRoundFlatIconButton method), 229
set_icon_color() (kivymd.uix.button.MDRectangleFlatButton method), 228
set_icon_color() (kivymd.uix.button.MDRoundFlatButton method), 229
set_item() (kivymd.uix.dropdownitem.MDDropDownItem method), 91
set_items_color() (kivymd.uix.navigationrail.MDNavigationRail method), 67
set_items_visible() (kivymd.uix.navigationrail.MDNavigationRail method), 67
set_left_action() (kivymd.uix.bannerMDBanner method), 278
set_md_bg_color() (kivymd.uix.button.MDFillRoundFlatButton method), 229
set_md_bg_color() (kivymd.uix.button.MDFloatingActionButton method), 230
set_md_bg_color() (kivymd.uix.toolbar.MDToolbar method), 76
set_menu_properties() (kivymd.menu.MDDropdownMenu method), 40
set_month_day() (kivymd.uix.picker.MDDatePicker method), 159
set_notch() (kivymd.uix.toolbar.MDToolbar method), 76
set_objects_labels() (kivymd.uix.textfield.MDTextField method), 259
set_paddings() (kivymd.uix.expansionpanel.MDExpansionPanelLabel attribute), 218
set_pos_bottom_buttons() (kivymd.uix.button.MDFloatingActionButtonSpeedDial method), 233
set_pos_labels() (kivymd.uix.button.MDFloatingActionButton method), 233
set_pos_root_button() (kivymd.uix.button.MDFloatingActionButtonSpeedDial method), 233
set_position_to_current_year() (kivymd.uix.picker.MDDatePicker method), 159
set_right_action() (kivymd.uix.bannerMDBanner method), 278
set_selected_widget() (kivymd.uix.picker.MDDatePicker method), 159
set_shadow() (kivymd.uix.toolbar.MDToolbar method), 76
set_size() (kivymd.uix.button.MDFloatingActionButton method), 230
set_size() (kivymd.uix.button.MDIconButton method), 230
set_spinner() (kivymd.uix.refreshlayout.RefreshSpinner method), 230
set_state() (kivymd.uix.navigationdrawer.MDNavigationDrawer method), 131
set_term_vel() (kivymd.effects.roulettescroll.RouletteScrollEffect method), 326
set_text() (kivymd.uix.textfield.MDTextField method), 259
set_text_color() (kivymd.uix.button.MDFillRoundFlatButton method), 229
set_text_color() (kivymd.uix.button.MDFillRoundFlatButton method), 229
set_text_full_date() (kivymd.uix.picker.MDDatePicker method), 59
set_time() (kivymd.uix.picker.MDTimePicker method), 160
set_type_banner() (kivymd.uix.bannerMDBanner method), 278
set_width() (kivymd.uix.navigationrail.MDNavigationRail method), 67
shadow_group (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 296
shadow_pos (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 296
shadow_preset() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 298
shake() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 309
sheet_list (kivymd.uix.bottomsheet.MDListBottomSheet attribute), 218
shift_y (kivymd.uix.tooltip.MDTooltip attribute), 89
show() (kivymd.uix.bannerMDBanner method), 278
Show() (kivymd.uix.filemanager.MDFileManager method), 126
Show_Skip() (kivymd.uix.filemanager.MDFileManager method), 126
show_error() (kivymd.utils.hot_reload_viewer.HotReloadViewer method), 343
show_hidden_files (kivymd.uix.filemanager.MDFileManager attribute), 126
show_off (kivymd.uix.slider.MDSlider attribute), 247
shrink() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 309
size_duration (kivymd.uix.swiper.MDSwiper attribute), 45
size_transition (kivymd.uix.swiper.MDSwiper attribute), 45
sleep() (in module kivymd.utils.asynckivy), 337
SmartTile (class in kivymd.uix.imagelist), 191

SmartTileWithLabel (*class in kivymd.uix.imagelist*), 192
SmartTileWithStar (*class in kivymd.uix.imagelist*), 192
Snackbar (*class in kivymd.uix.snackbar*), 244
soft_shadow_c1 (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* attribute), 296
soft_shadow_offset (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* attribute), 297
soft_shadow_pos (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* attribute), 296
soft_shadow_size (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* attribute), 296
sort_by (*kivymd.uix.filemanager.MDFileManager* attribute), 126
sort_by_desc (*kivymd.uix.filemanager.MDFileManager* attribute), 126
sorted_on (*kivymd.uix.datatables.MDDDataTable* attribute), 55
sorted_order (*kivymd.uix.datatables.MDDDataTable* attribute), 55
source (*kivymd.uix.bottomsheet.GridBottomSheetItem* attribute), 218
source (*kivymd.uix.imagelist.SmartTile* attribute), 192
source (*kivymd.uix.label.MDIcon* attribute), 165
source (*kivymd.utils.fitimage.FitImage* attribute), 340
specific_secondary_text_color (*kivymd.uix.behaviors.backgroundcolor_behavior*.*SpecificBackgroundColorBehavior* attribute), 312
specific_text_color (*kivymd.uix.behaviors.backgroundcolor_behavior*.*SpecificBackgroundColorBehavior* attribute), 312
SpecificBackgroundColorBehavior (*class in kivymd.uix.behaviors.backgroundcolor_behavior*), 312
spinner_color (*kivymd.uix.refreshlayout.RefreshSpinner* attribute), 131
standard_increment (*kivymd.theming.ThemeManager* attribute), 17
stars (*kivymd.uix.imagelist.SmartTileWithStar* attribute), 192
start() (*in module kivymd.utils.asynckivy*), 337
start() (*kivymd.effects.roulettescroll.RouletteScrollEffect* method), 326
start() (*kivymd.effects.stiffscroll.StiffScrollView* method), 327
start() (*kivymd.uix.progressbar.MDProgressBar* method), 122
start() (*kivymd.uix.taptargetview.MDTapTargetView* method), 178
start() (*kivymd.utils.fpsmonitor.FpsMonitor* method), 341
start_anim_spinner() (*kivymd.uix.refreshlayout.RefreshSpinner* method), 131
start_from (*kivymd.uix.circularlayout.MDCircularLayout* attribute), 128
start_ripple() (*kivymd.uix.behaviors.ripple_behavior.CommonRipple* method), 304
state (*kivymd.uix.button.MDFloatingActionButtonSpeedDial* attribute), 232
state (*kivymd.uix.card.MDCardSwipe* attribute), 209
state (*kivymd.uix.navigationdrawer.MDNavigationDrawer* attribute), 138
state (*kivymd.uix.taptargetview.MDTapTargetView* attribute), 178
status (*kivymd.uix.navigationdrawer.MDNavigationDrawer* attribute), 138
StiffScrollView (*class in kivymd.effects.stiffscroll*), 327
stop() (*kivymd.effects.stiffscroll.StiffScrollView* method), 328
stop() (*kivymd.uix.progressbar.MDProgressBar* method), 122
stop() (*kivymd.uix.taptargetview.MDTapTargetView* method), 178
stop_on_outer_touch (*kivymd.uix.taptargetview.MDTapTargetView* attribute), 178
stop_on_target_touch (*kivymd.uix.taptargetview.MDTapTargetView* attribute), 178
swipe_distance (*kivymd.uix.card.MDCardSwipe* attribute), 208
swipe_distance (*kivymd.uix.navigationdrawer.MDNavigationDrawer* attribute), 139
swipe_distance (*kivymd.uix.swiper.MDSwiper* attribute), 45
swipe_edge_width (*kivymd.uix.navigationdrawer.MDNavigationDrawer* attribute), 139
swipe_left() (*kivymd.uix.swiper.MDSwiper* method), 46
swipe_on_scroll (*kivymd.uix.swiper.MDSwiper* attribute), 45
swipe_right() (*kivymd.uix.swiper.MDSwiper* method), 46
swipe_transition (*kivymd.uix.swiper.MDSwiper* attribute), 45
switch_tab() (*kivymd.uix.bottomnavigation.MDBottomNavigation* method), 273
switch_tab() (*kivymd.uix.tab.MDTabs* method), 117
sync_theme_styles() (*kivymd.theming.ThemeManager* method), 19

T

tab_bar_height (*kivymd.uix.tab.MDTabs* attribute), 115

tab_header (*kivymd.uix.bottomnavigation.MDBottomNavigation*.attribute), 272
tab_hint_x (*kivymd.uix.tab.MDTabs* attribute), 115
tab_indicator_anim (*kivymd.uix.tab.MDTabs* attribute), 115
tab_indicator_height (*kivymd.uix.tab.MDTabs* attribute), 115
tab_indicator_type (*kivymd.uix.tab.MDTabs* attribute), 115
tab_label (*kivymd.uix.tab.MDTabsBase* attribute), 114
tab_label_font_style (*kivymd.uix.tab.MDTabsBase* attribute), 114
tab_label_text (*kivymd.uix.tab.MDTabsBase* attribute), 114
tab_padding (*kivymd.uix.tab.MDTabs* attribute), 115
TabbedPanelBase (class in *kivymd.uix.bottomnavigation*), 272
tabs (*kivymd.uix.bottomnavigation.TabbedPanelBase* attribute), 272
target_circle_color (*kivymd.uix.taptargetview.MDTapTargetView* attribute), 176
target_radius (*kivymd.uix.taptargetview.MDTapTargetView* attribute), 176
target_widget (*kivymd.effects.stiffscroll.StiffScrollEffect* attribute), 327
temp_font_path (in module *kivymd.tools.release.update_icons*), 335
temp_path (in module *kivymd.tools.release.update_icons*), 335
temp_preview_path (in module *kivymd.tools.release.update_icons*), 335
temp_repo_path (in module *kivymd.tools.release.update_icons*), 335
terminal_velocity (*kivymd.effects.roulettescroll.Roulette*.attribute), 325
text (*kivymd.uix.bannerMDBanner* attribute), 277
text (*kivymd.uix.bottomnavigation.MDTab* attribute), 272
text (*kivymd.uix.chip.MDChip* attribute), 83
text (*kivymd.uix.dialog.MDDialog* attribute), 181
text (*kivymd.uix.dropdownitem.MDDropDownItem* attribute), 91
text (*kivymd.uix.imagelist.SmartTileWithLabel* attribute), 192
text (*kivymd.uix.label.MDLabel* attribute), 165
text (*kivymd.uix.snackbar.Snackbar* attribute), 244
text (*kivymd.uix.tab.MDTabsBase* attribute), 114
text (*kivymd.utils.hot_reload_viewer.HotReloadErrorText* attribute), 342
text_button_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 153
text_color (*kivymd.theming.ThemeManager* attribute), 16

text_color (*kivymd.uix.chip.MDChip* attribute), 84
text_color (*kivymd.uix.label.MDLabel* attribute), 165
text_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 151
text_color (*kivymd.uix.textfield.MDTextField* attribute), 258
text_color_active (*kivymd.uix.bottomnavigation.MDBottomNavigation* attribute), 273
text_color_active (*kivymd.uix.tab.MDTabs* attribute), 116
text_color_normal (*kivymd.uix.bottomnavigation.MDBottomNavigation* attribute), 272
text_color_normal (*kivymd.uix.tab.MDTabs* attribute), 116
text_colors (in module *kivymd.color_definitions*), 24
text_current_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 152
text_title (*kivymd.uix.navigationrail.MDNavigationRail* attribute), 64
text_toolbar_color (*kivymd.uix.picker.BaseDialogPicker* attribute), 150
text_weekday_color (*kivymd.uix.picker.MDDatePicker* attribute), 157
ThemableBehavior (class in *kivymd.theming*), 19
theme_cls (*kivymd.app.MDApp* attribute), 21
theme_cls (*kivymd.theming.ThemableBehavior* attribute), 19
theme_colors (in module *kivymd.color_definitions*), 25
theme_font_styles (in module *kivymd.font_definitions*), 28
theme_style (*kivymd.theming.ThemeManager* attribute), 13
theme_text_color (*kivymd.uix.button.MDRaisedButton* attribute), 228
theme_text_color (*kivymd.uix.label.MDLabel* attribute), 165
theme_thumb_color (*kivymd.uix.selectioncontrol.MDSwitch* attribute), 197
theme_thumb_down_color (*kivymd.uix.selectioncontrol.MDSwitch* attribute), 197
ThemeManager (class in *kivymd.theming*), 9
ThreeLineAvatarIconListItem (class in *kivymd.uix.list*), 103
ThreeLineAvatarListItem (class in *kivymd.uix.list*), 103
ThreeLineIconListItem (class in *kivymd.uix.list*), 103
ThreeLineList ListItem (class in *kivymd.uix.list*), 103
ThreeLineRightIconListItem (class in *kivymd.uix.list*), 103
thumb_color (*kivymd.uix.selectioncontrol.MDSwitch* attribute), 197
thumb_color_disabled (*kivymd.uix.selectioncontrol.MDSwitch* attribute), 197

tribute), 197
thumb_color_down (kivymd.uix.selectioncontrol.MDSwitch attribute), 197
tile_text_color (kivymd.uix.imagelist.SmartTileWithLabel attribute), 192
time (kivymd.uix.picker.MDTimePicker attribute), 160
title (kivymd.uix.backdrop.MDBackdrop attribute), 79
title (kivymd.uix.dialog.MDDialog attribute), 180
title (kivymd.uix.picker.BaseDialogPicker attribute), 147
title (kivymd.uix.tab.MDTabsBase attribute), 114
title (kivymd.uix.toolbar.MDToolbar attribute), 75
title_icon_mode (kivymd.uix.tab.MDTabs attribute), 117
title_icon_mode (kivymd.uix.tab.MDTabsBase attribute), 114
title_input (kivymd.uix.picker.BaseDialogPicker attribute), 147
title_is_capital (kivymd.uix.tab.MDTabsBase attribute), 114
title_position (kivymd.uix.taptargetview.MDTapTargetView attribute), 178
title_text (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
title_text_bold (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
title_text_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
title_text_size (kivymd.uix.taptargetview.MDTapTargetView attribute), 177
Toast (class in kivymd.toast.kivytoast.kivytoast), 330
toast () (in module kivymd.toast.androidtoast.androidtoast), 329
toast () (in module kivymd.toast.kivytoast.kivytoast), 331
toast () (kivymd.toast.kivytoast.kivytoast.Toast method), 331
tooltip_bg_color (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_bg_color (kivymd.uix.tooltip.MDTooltipViewClass attribute), 89
tooltip_display_delay (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_font_style (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_font_style (kivymd.uix.tooltip.MDTooltipViewClass attribute), 89
tooltip_radius (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_radius (kivymd.uix.tooltip.MDTooltipViewClass attribute), 89
tooltip_text (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_text (kivymd.uix.tooltip.MDTooltipViewClass attribute), 89
tooltip_text_color (kivymd.uix.tooltip.MDTooltip attribute), 88
tooltip_text_color (kivymd.uix.tooltip.MDTooltipViewClass attribute), 89
TOUCH_TARGET_HEIGHT (in module kivymd.material_resources), 323
TouchBehavior (class in kivymd.uix.behaviors.touch_behavior), 310
transformation_from_dialog_input_date () (kivymd.uix.picker.MDDatePicker method), 158
transformation_from_dialog_select_year () (kivymd.uix.picker.MDDatePicker method), 158
transformation_to_dialog_input_date () (kivymd.uix.picker.MDDatePicker method), 158
transformation_to_dialog_select_year () (kivymd.uix.picker.MDDatePicker method), 158
transition_duration (kivymd.uix.swiper.MDSwiper attribute), 45
transition_max (kivymd.effects.stiffscroll.StiffScrollView attribute), 327
transition_min (kivymd.effects.stiffscroll.StiffScrollView attribute), 327
toast () (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 309
TwoLineAvatarIconListItem (class in kivymd.uix.list), 103
TwoLineAvatarListItem (class in kivymd.uix.list), 103
TwoLineIconListItem (class in kivymd.uix.list), 103
TwoLineListIcon (class in kivymd.uix.list), 103
TwoLineRightIconListItem (class in kivymd.uix.list), 103
type (kivymd.uix.banner.MDBanner attribute), 277
type (kivymd.uix.dialog.MDDialog attribute), 186
type (kivymd.uix.navigationdrawer.MDNavigationDrawer attribute), 138
type (kivymd.uix.progressbar.MDProgressBar attribute), 122
type (kivymd.uix.toolbar.MDToolbar attribute), 75
type_swipe (kivymd.uix.card.MDCardSwipe attribute), 209
underline_color (kivymd.uix.tab.MDTabs attribute), 116
unfocus_color (kivymd.uix.behaviors.focus_behavior.FocusBehavior attribute), 302
unselected_all () (kivymd.uix.selection.MDSelectionList method), 284

unselected_color (*kivymd.uix.selectioncontrol.MDCheckbox* attribute), 196
unzip_archive() (in module *kivymd.tools.release.update_icons*), 336
update() (*kivymd.effects.stiffscroll.StiffScrollEffect* method), 327
update() (*kivymd.utils.hot_reload_viewer.HotReloadViewer* method), 343
update_action_bar() (*kivymd.uix.toolbar.MDToolbar* method), 76
update_action_bar_text_colors() (*kivymd.uix.toolbar.MDToolbar* method), 76
update_background_origin() (*kivymd.uix.behaviors.backgroundcolor_behavior*.method), 312
update_calendar() (*kivymd.uix.picker.MDDatePicker* method), 159
update_calendar_for_date_range() (*kivymd.uix.picker.MDDatePicker* method), 158
update_color() (*kivymd.uix.selectioncontrol.MDCheckbox* method), 196
update_font_style() (*kivymd.uix.label.MDLabel* method), 165
update_fps() (*kivymd.utils.fpsmonitor.FpsMonitor* method), 341
update_group_property() (*kivymd.uix.behaviors.elevation.CommonElevation*.method), 298
update_height() (*kivymd.uix.dialog.MDDialog* method), 188
update_icon() (*kivymd.uix.selectioncontrol.MDCheckbox* method), 196
update_icon_color() (*kivymd.button.MDFillRoundFlatButton* method), 229
update_icon_color() (*kivymd.uix.tab.MDTabs* method), 117
update_icons() (in module *kivymd.tools.release.update_icons*), 336
update_init_py() (in module *kivymd.tools.release.make_release*), 335
update_label_text() (*kivymd.uix.tab.MDTabsBase* method), 115
update_md_bg_color() (*kivymd.button.MDFillRoundFlatButton* method), 229
update_md_bg_color() (*kivymd.button.MDIconButton* method), 230
update_md_bg_color() (in module *kivymd.button.MDRectangleFlatIconButton* method), 228
update_md_bg_color() (in module *kivymd.button.MDRoundFlatButton* method), 229
update_md_bg_color() (in module *kivymd.uix.card.MDCard* method), 208
update_md_bg_color() (in module *kivymd.uix.toolbar.MDToolbar* method), 76
update_opposite_colors() (*kivymd.uix.toolbar.MDToolbar* method), 76
update_primary_color() (*kivymd.uix.selectioncontrol.MDCheckbox* method), 196
update_readme() (in module *kivymd.tools.release.make_release*), 335
update_row_data() (*kivymd.uix.datatables.MDDDataTable* method), 59
update_scrim_rectangle() (*kivymd.uix.navigationdrawer.MDNavigationLayout* method), 137
update_status() (*kivymd.uix.navigationdrawer.MDNumericUpDown* method), 139
update_text_color() (*kivymd.button.MDFillRoundFlatButton* method), 229
update_text_color() (*kivymd.button.MDFloatingActionButton* method), 230
update_text_color() (*kivymd.button.MDRaisedButton* method), 228
update_text_full_date() (*kivymd.picker.MDDatePicker* method), 159
update_velocity() (*kivymd.effects.roulettescroll.RouletteScrollView* method), 326
update_velocity() (*kivymd.effects.stiffscroll.StiffScrollView* method), 327
update_width() (*kivymd.uix.dialog.MDDialog* method), 188
updated_interval (*kivymd.utils.fpsmonitor.FpsMonitor* attribute), 341
url (in module *kivymd.tools.release.update_icons*), 335
use_access (*kivymd.uix.filemanager.MDFileManager* attribute), 126
use_action_button (*kivymd.uix.navigationrail.MDNavigationRail* attribute), 64
use_hover_behavior (*kivymd.uix.navigationrail.MDNavigationRail*

attribute), 62
use_pagination (*kivymd.uix.datatables.MDDDataTable*
attribute), 55
use_resizeable (*kivymd.uix.navigationrail.MDNavigationRail*
attribute), 63
use_title (*kivymd.uix.navigationrail.MDNavigationRail*
attribute), 64

V

value_transparent (*kivymd.uix.bottomsheet.MDBottomSheet*
attribute), 217
ver_growth (*kivymd.uix.menu.MDDropdownMenu* at-
tribute), 39
vertical_pad (*kivymd.uix.bannerMDBanner* at-
tribute), 277
visible (*kivymd.uix.navigationrail.MDNavigationRail*
attribute), 65

W

widget (*kivymd.uix.taptargetview.MDTapTargetView* at-
tribute), 175
widget_position (*kivymd.uix.taptargetview.MDTapTargetView*
attribute), 178
widget_style (*kivymd.theming.ThemableBehavior* at-
tribute), 19
width_mult (*kivymd.uix.menu.MDDropdownMenu* at-
tribute), 39
width_mult (*kivymd.uix.swiper.MDSwiper* attribute), 45
width_offset (*kivymd.uix.dialog.MDDialog* attribute),
186
wobble() (*kivymd.uix.behaviors.magic_behavior.MagicBehavior*
method), 309

Y

year (*kivymd.uix.picker.MDDatePicker* attribute), 158