
KivyMD

Release 1.0.2

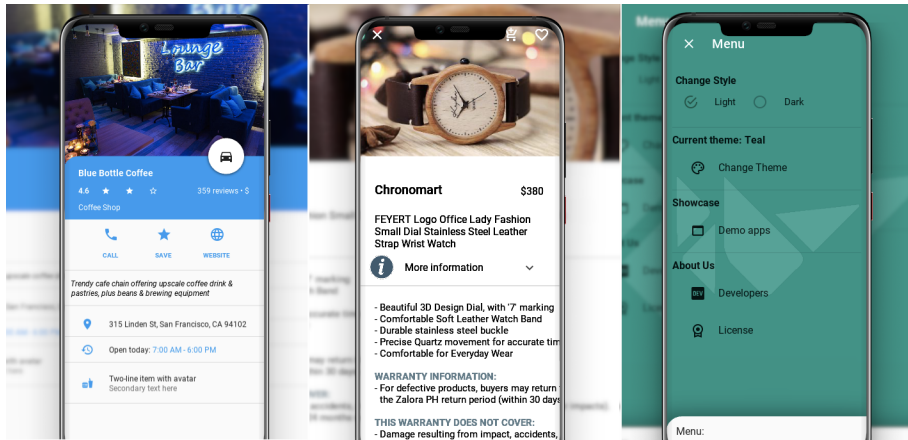
Andrés Rodríguez, Ivanov Yuri, Artem Bulgakov and KivyMD contributors

Aug 11, 2022

CONTENTS

1	KivyMD	1
2	Contents	3
2.1	Getting Started	3
2.2	Themes	7
2.3	Components	33
2.4	Controllers	416
2.5	Behaviors	417
2.6	Effects	453
2.7	Templates	459
2.8	Changelog	465
2.9	About	476
2.10	KivyMD	477
3	Indices and tables	511
	Python Module Index	513
	Index	515

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.

This library is a fork of the [KivyMD project](#). We found the strength and brought this project to a new level.

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
```

Note: 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 .
```

Note: 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")

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
from kivy.utils import get_color_from_hex
```

(continues on next page)

(continued from previous page)

```

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

<RectangleFlatButton>:
    ripple_color: 0, 0, 0, .2
    background_color: 0, 0, 0, 0
    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: get_color_from_hex("#0F0F0F")
        Rectangle:
            pos: self.pos
            size: self.size
"""

class RectangleFlatButton(TouchRippleBehavior, Button):
    primary_color = get_color_from_hex("#EB8933")

    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",

```

(continues on next page)

(continued from previous page)

```
        pos_hint={"center_x": 0.5, "center_y": 0.5},
        size_hint=(None, None),
        size=(dp(110), dp(35)),
        ripple_color=(0.8, 0.8, 0.8, 0.5),
    )
)
return screen
```

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

And the equivalent with *KivyMD*:

```
from kivymd.app import MDApp
from kivymd.ui.screen import MDScreen
from kivymd.ui.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"

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

MainApp().run()
```

KivyMD:

Kivy:

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 definition 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.

Imperative python style with KV

```
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": {
        "200": "#212121",
```

(continues on next page)

(continued from previous page)

```

        "500": "#212121",
        "700": "#212121",
    },
    "Red": {
        "200": "#C25554",
        "500": "#C25554",
        "700": "#C25554",
    },
    "Light": {
        "StatusBar": "E0E0E0",
        "AppBar": "#202020",
        "Background": "#2E3032",
        "CardsDialogs": "FFFFFF",
        "FlatButtonDown": "CCCCCC",
    },
}

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

    MDTopAppBar:
        title: "Custom theme"

    MDTabs:
        id: tabs

<Tab>

    MDIconButton:
        id: icon
        icon: root.icon
        icon_size: "48sp"
        theme_icon_color: "Custom"
        icon_color: "white"
        pos_hint: {"center_x": .5, "center_y": .5}
'''

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

```

(continues on next page)

(continued from previous page)

```

self.theme_cls.primary_palette = "Teal"
self.theme_cls.accent_palette = "Red"
return Builder.load_string(KV)

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

```

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

Declarative python style

```

from kivy.properties import ObjectProperty

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDIconButton
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.tab import MDTabsBase, MDTabs
from kivymd.icon_definitions import md_icons
from kivymd.uix.toolbar import MDTopAppBar

colors = {
    "Teal": {
        "200": "#212121",
        "500": "#212121",
        "700": "#212121",
    },
    "Red": {
        "200": "#C25554",
        "500": "#C25554",
        "700": "#C25554",
    },
    "Light": {
        "StatusBar": "E0E0E0",
        "AppBar": "#202020",
        "Background": "#2E3032",
        "CardsDialogs": "FFFFFF",
        "FlatButtonDown": "CCCCCC",
    },
}

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

    icon = ObjectProperty()

class Example(MDApp):

```

(continues on next page)

(continued from previous page)

```

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

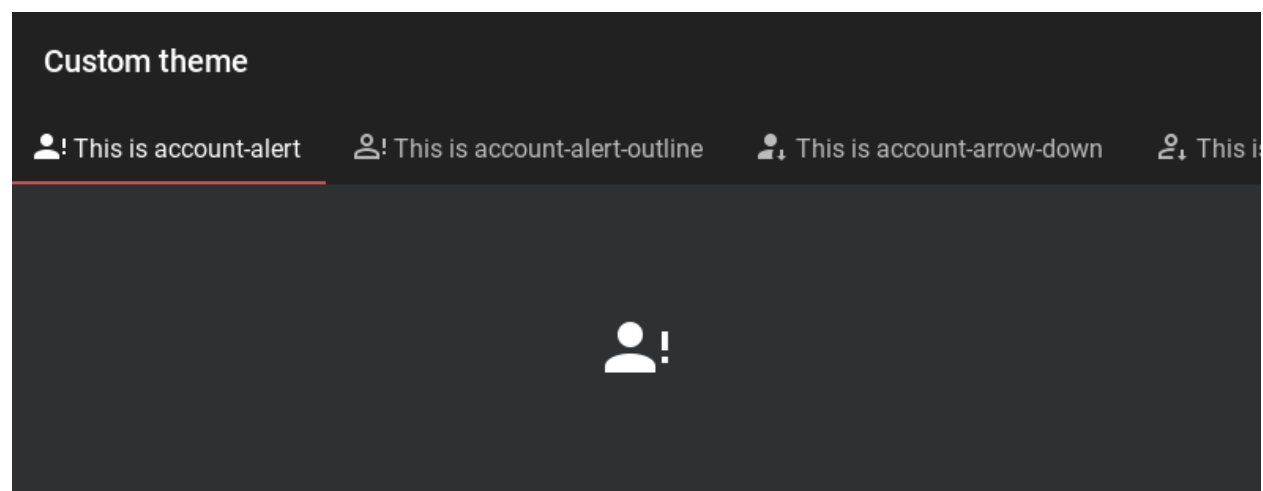
def build(self):
    self.theme_cls.colors = colors
    self.theme_cls.primary_palette = "Teal"
    self.theme_cls.accent_palette = "Red"

    return (
        MDBoxLayout(
            MDTopAppBar(title="Custom theme"),
            MDTabs(id="tabs",
                orientation="vertical",
            )
        )
    )

def on_start(self):
    for name_tab in self.icons:
        self.root.ids.tabs.add_widget(
            Tab(
                MDIconButton(
                    icon=name_tab,
                    icon_size="48sp",
                    theme_icon_color="Custom",
                    icon_color="white",
                    pos_hint={"center_x": .5, "center_y": .5},
                ),
                title="This is " + name_tab,
                icon=name_tab,
            )
        )

```

Example().run()



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

Warning: Please note that the key 'Red' is a required key for the dictionary `kivymd.color_definition.colors`.

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:

Imperative python style with KV

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDRectangleFlatButton:
        text: "Hello, World"
        pos_hint: {"center_x": .5, "center_y": .5}
...

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Red" # "Purple", "Red"

        return Builder.load_string(KV)

Example().run()
```

Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange" # "Purple", "Red"
```

(continues on next page)

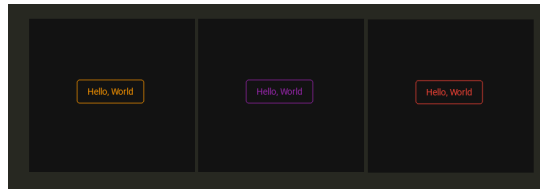
(continued from previous page)

```

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

```

Example().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:

Imperative python style with KV

```

from kivymd.app import MDAApp
from kivymd.ui.screen import MDScreen
from kivymd.ui.button import MDRectangleFlatButton

class MainApp(MDAApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.primary_hue = "200" # "500"
        screen = MDScreen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
        return screen

```

MainApp().run()

Declarative python style

```

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

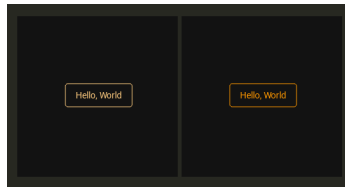
class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_hue = "200" # "500"

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

Example().run()

```

With a value of `self.theme_cls.primary_hue = "200"` and `self.theme_cls.primary_hue = "500"`:



`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.

`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 lighter color).

Declarative style with KV

```

from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    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 = "Orange"
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

MainApp().run()

```

Declarative python style

```

from kivymd.app import MDApp
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"

        return (
            MDScreen(
                MDRaisedButton(
                    text="Primary light",
                    pos_hint={"center_x": 0.5, "center_y": 0.7},
                    md_bg_color=self.theme_cls.primary_light,
                ),
                MDRaisedButton(

```

(continues on next page)

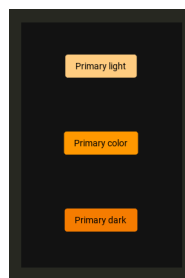
(continued from previous page)

```

        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=self.theme_cls.primary_dark,
    ),
)
)

```

```
Example().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 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. See `kivymd.uix.tab.MDTabsBar.indicator_color` attribute.

`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.

material_style

Material design style. Available options are: 'M2', 'M3'.

New in version 1.0.0.

See also:

Material Design 2 and *Material Design 3*

material_style is an *OptionProperty* and defaults to 'M2'.

theme_style

App theme style.

Imperative python style

```
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
from kivymd.uix.button import MDRectangleFlatButton

class MainApp(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark" # "Light"
        screen = MDScreen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
        return screen

MainApp().run()
```

Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.button import MDRectangleFlatButton
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
```

(continues on next page)

(continued from previous page)

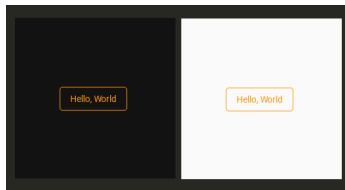
```

self.theme_cls.primary_palette = "Orange"
self.theme_cls.theme_style = "Dark" # "Light"

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

```

```
Example().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`.

Declarative style with KV

```

from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDBoxLayout:
    MDWidget:
        md_bg_color: app.theme_cls.bg_light

    MDBoxLayout:
        md_bg_color: app.theme_cls.bg_normal

    MDBoxLayout:
        md_bg_color: app.theme_cls.bg_dark

    MDBoxLayout:
        md_bg_color: app.theme_cls.bg_darkest
'''

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

```

(continues on next page)

(continued from previous page)

```
return Builder.load_string(KV)
```

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

Declarative python style

```
from kivymd.app import MDApp
from kivymd.ui.boxlayout import MDBoxLayout
from kivymd.ui.widget import MDWidget

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

        return (
            MDBoxLayout(
                MDWidget(
                    md_bg_color=self.theme_cls.bg_light,
                ),
                MDWidget(
                    md_bg_color=self.theme_cls.bg_normal,
                ),
                MDWidget(
                    md_bg_color=self.theme_cls.bg_dark,
                ),
                MDWidget(
                    md_bg_color=self.theme_cls.bg_darkest,
                ),
            )
        )

Example().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.

disabled_primary_color

The greyscale disabled version of the current application theme color in rgba format.

New in version 1.0.0.

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

opposite_disabled_primary_color

The opposite value of color in the *disabled_primary_color*.

New in version 1.0.0.

opposite_disabled_primary_color is an *AliasProperty* that returns the value in rgba format for *opposite_disabled_primary_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 MDTextField.

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.

Declarative style with KV

```
from kivy.core.text import LabelBase
from kivy.lang import Builder

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

KV = '''
MDScreen:

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

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

        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)
```

(continues on next page)

(continued from previous page)

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

Declarative python style

```
from kivy.core.text import LabelBase

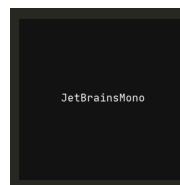
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
from kivymd.uix.label import MDLabel
from kivymd.font_definitions import theme_font_styles

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

        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 (
            MDScreen(
                MDLabel(
                    text="JetBrainsMono",
                    halign="center",
                    font_style="JetBrainsMono",
                )
            )
        )

MainApp().run()
```



`font_styles` is an `DictProperty`.

`on_theme_style(self, interval: int, theme_style: str)`

`set_clearcolor_by_theme_style(self, theme_style)`

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

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.

Imperative python style

```
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
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 = MDScreen()
        screen.add_widget(
            MDRectangleFlatButton(
                text="Hello, World",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )
        return screen
```

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

Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
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"
        )
```

(continues on next page)

(continued from previous page)

```

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

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'*:

```

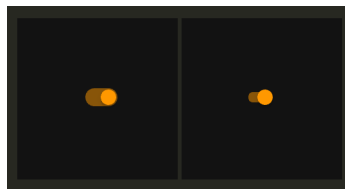
MDSwitch:
    widget_style: "ios"

```

```

MDSwitch:
    widget_style: "android"

```



widget_style is an *OptionProperty* and defaults to *'android'*.

opposite_colors

For some widgets, for example, for a widget *MDTopAppBar* changes the color of the label to the color opposite to the main theme.

```

MDTopAppBar:
    title: "MDTopAppBar"
    opposite_colors: True

```

MDToolbar


```
MDTopAppBar:
    title: "MDTopAppBar"
    opposite_colors: True
```

MDToolbar

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 = '''
MDScreen:

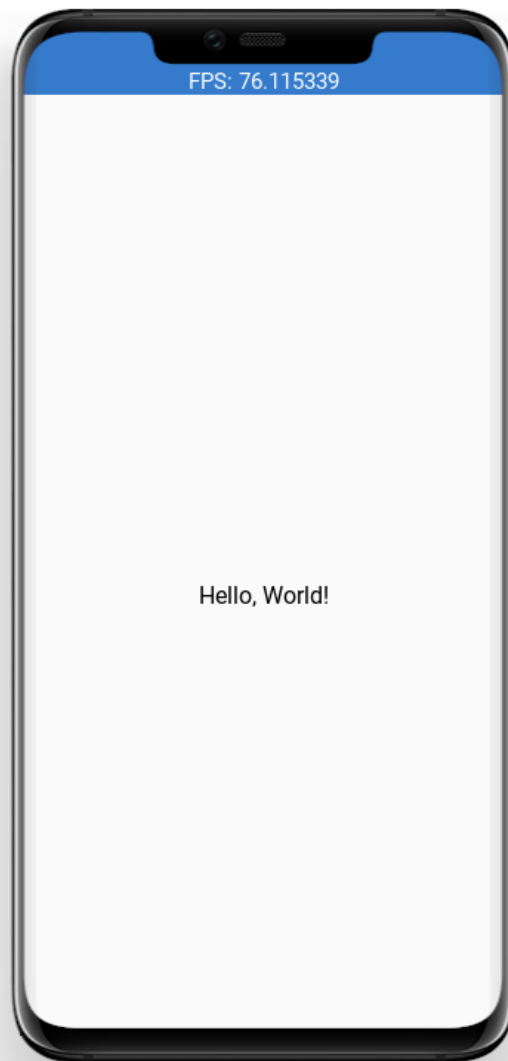
    MDLabel:
        text: "Hello, World!"
        halign: "center"
'''

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 [App](#) class documentation for more information.

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`.

`load_all_kv_files(self, path_to_directory: str)`

Recursively loads KV files from the selected directory.

New in version 1.0.0.

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.properties import ListProperty, StringProperty

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

demo = '''
<Root@MDBoxLayout>
    orientation: 'vertical'

    MDTopAppBar:
        title: app.title

    MDTabs:
        id: android_tabs
        on_tab_switch: app.on_tab_switch(*args)
```

(continues on next page)

(continued from previous page)

```

        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(MDBoxLayout, MDTabsBase):
    pass

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

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

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

        for name_tab in colors.keys():
            tab = Tab(text=name_tab)

```

(continues on next page)

(continued from previous page)

```

        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": 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"

```

(continues on next page)

(continued from previous page)

```

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']

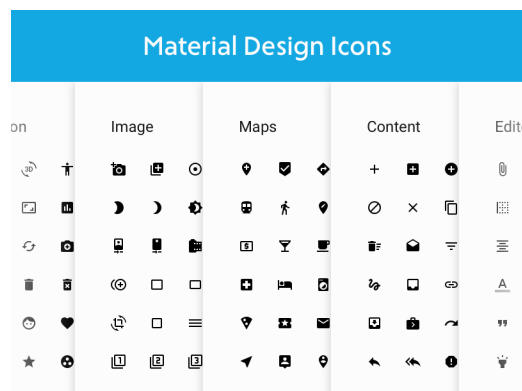
```

Valid theme colors.

2.2.4 Icon Definitions

See also:

Material Design Icons



List of icons from materialdesignicons.com. These expanded material design icons are maintained by Austin Andrews (Templarion on Github).

LAST UPDATED: Version 7.0.96

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

<CustomOneLineIconListItem>

    IconLeftWidget:

```

(continues on next page)

(continued from previous page)

```

        icon: root.icon

<PreviousMDIcons>

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

        MDBoxLayout:
            adaptive_height: True

            MDIconButton:
                icon: 'magnify'

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

        RecyclerView:
            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):
        self.ids.rv.data.append(
            {
                "viewclass": "CustomOneLineIconListItem",
                "icon": name_icon,
                "text": name_icon,

```

(continues on next page)

(continued from previous page)

```
        "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 AnchorLayout

New in version 1.0.0.

[AnchorLayout](#) class equivalent. Simplifies working with some widget properties. For example:

AnchorLayout

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

MDAnchorLayout

```
MDAnchorLayout:
    md_bg_color: app.theme_cls.primary_color
```

API - kivymd.uix.anchorlayout

`class kivymd.uix.anchorlayout.MDAnchorLayout(*args, **kwargs)`

Anchor layout class. For more information, see in the [AnchorLayout](#) class documentation.

2.3.2 Widget

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

Widget

```
Widget:
    size_hint: .5, None
    height: self.width

    canvas:
        Color:
            rgba: app.theme_cls.primary_color
        RoundedRectangle:
            pos: self.pos
            size: self.size
            radius: [self.height / 2,]
```

MDWidget

```

MDWidget:
    size_hint: .5, None
    height: self.width
    radius: self.height / 2
    md_bg_color: app.theme_cls.primary_color

```

API - kivymd.uix.widget

class kivymd.uix.widget.**MDWidget**(*args, **kwargs)

See Widget class documentation for more information.

New in version 1.0.0.

2.3.3 RecycleGridLayout

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

RecycleGridLayout

```

RecycleGridLayout:
    size_hint_y: None
    height: self.minimum_height

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

```

MDRecycleGridLayout

```

MDRecycleGridLayout:
    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.recyclegridlayout`

class `kivymd.uix.recyclegridlayout.MDRecycleGridLayout(*args, **kwargs)`

Recycle grid layout layout class. For more information, see in the [RecycleGridLayout](#) class documentation.

2.3.4 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

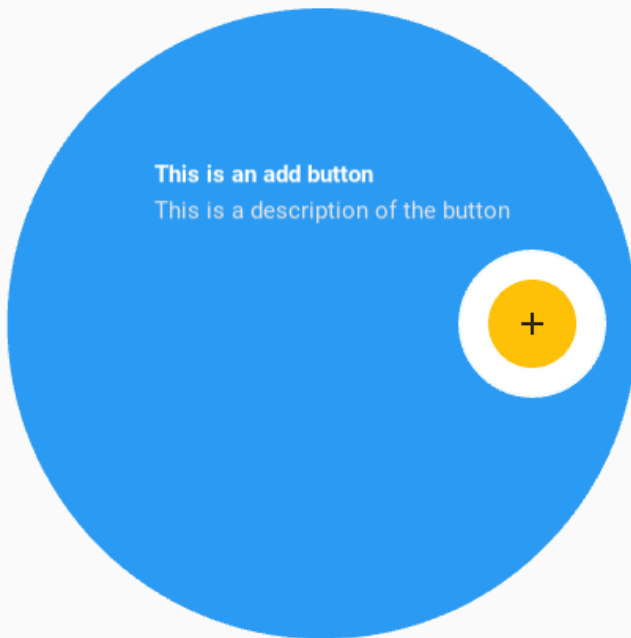
    def tap_target_start(self):
        if self.tap_target_view.state == "close":
            self.tap_target_view.start()
        else:
            self.tap_target_view.stop()

TapTargetViewDemo().run()
```

Widget position

Sets the position of the widget relative to the floating circle.

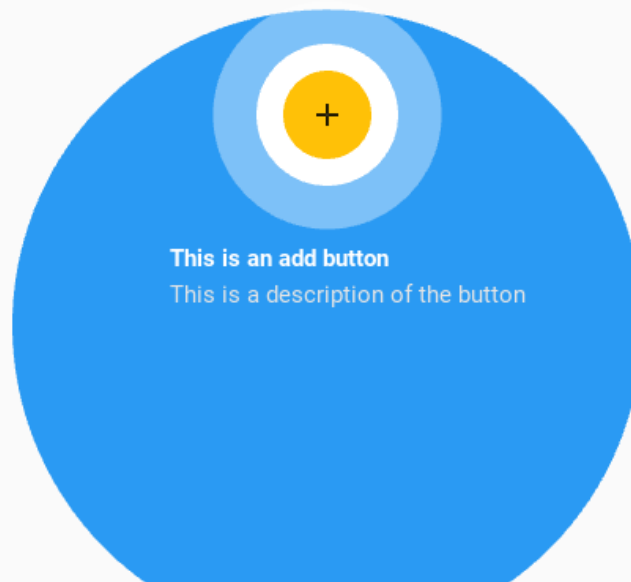
```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="right",  
)
```



```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="left",  
)
```



```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="top",  
)
```

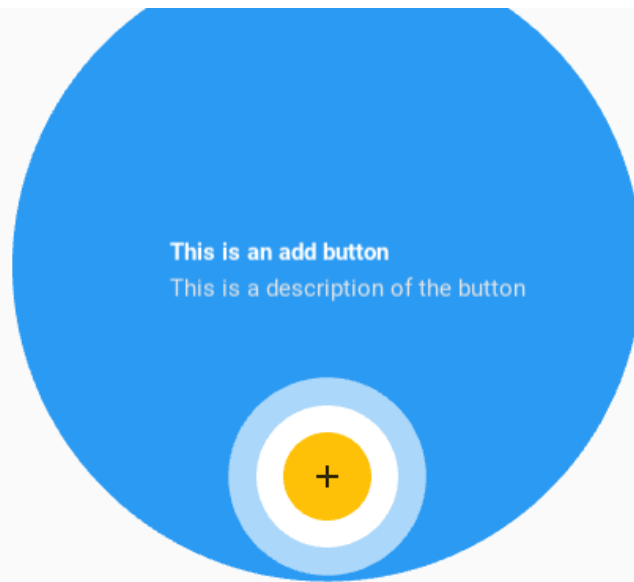


```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="bottom",
```

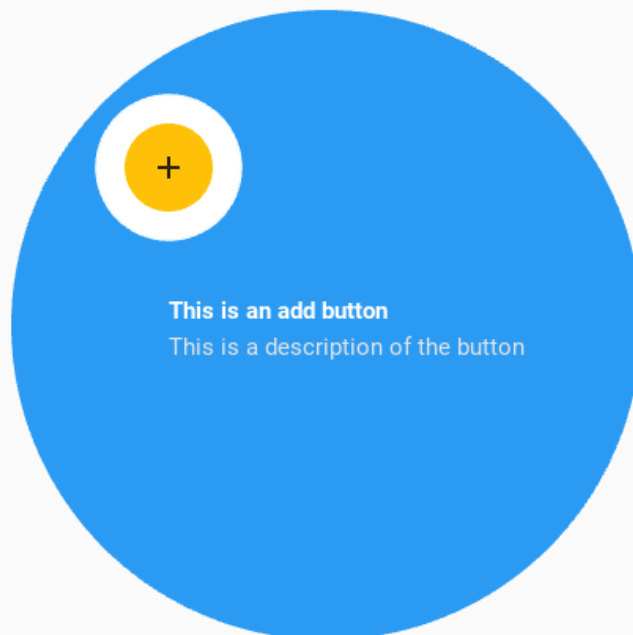
(continues on next page)

(continued from previous page)

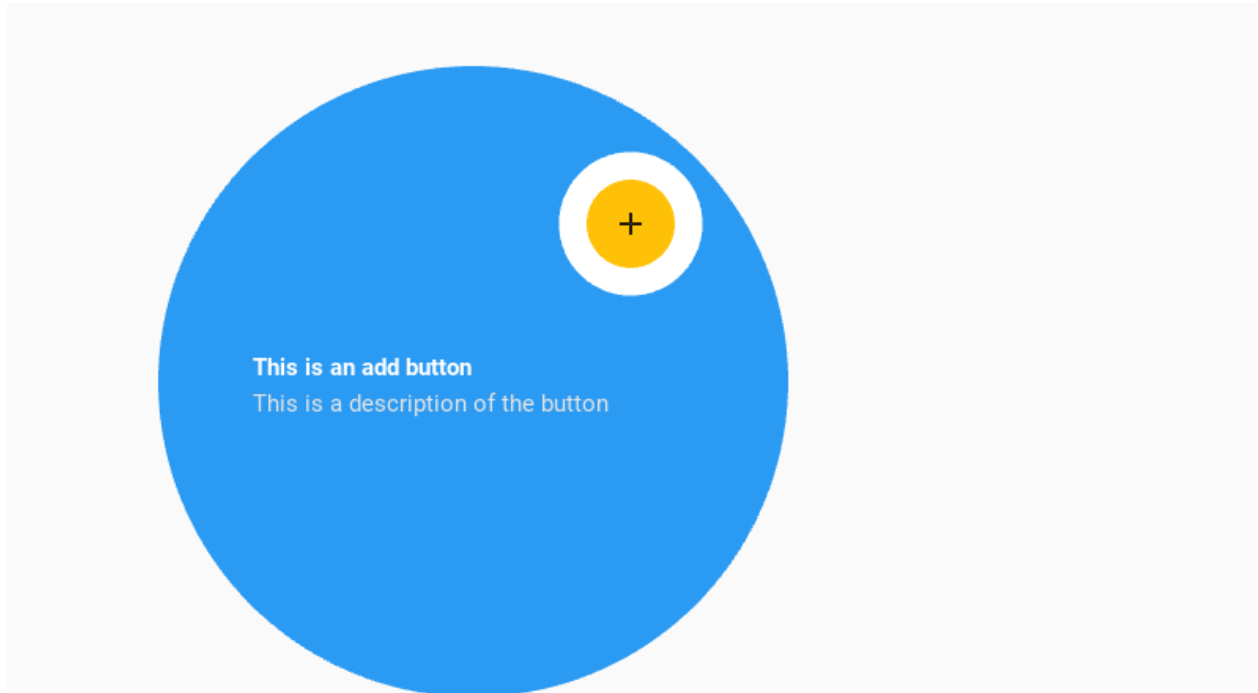
)



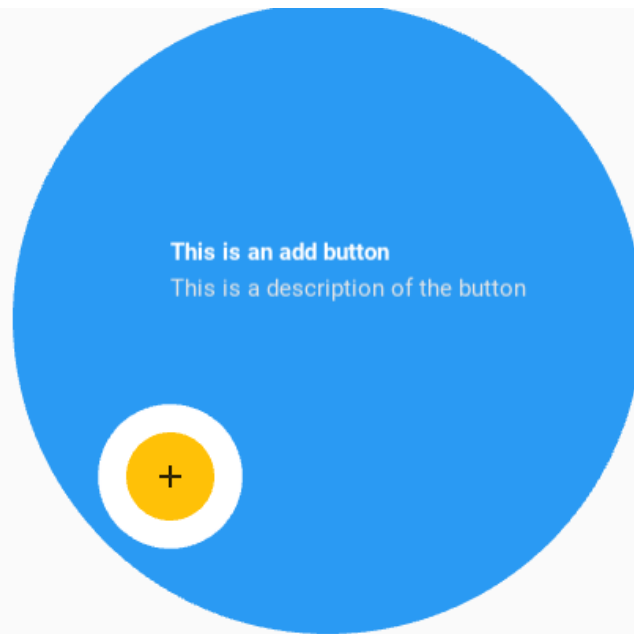
```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="left_top",  
)
```



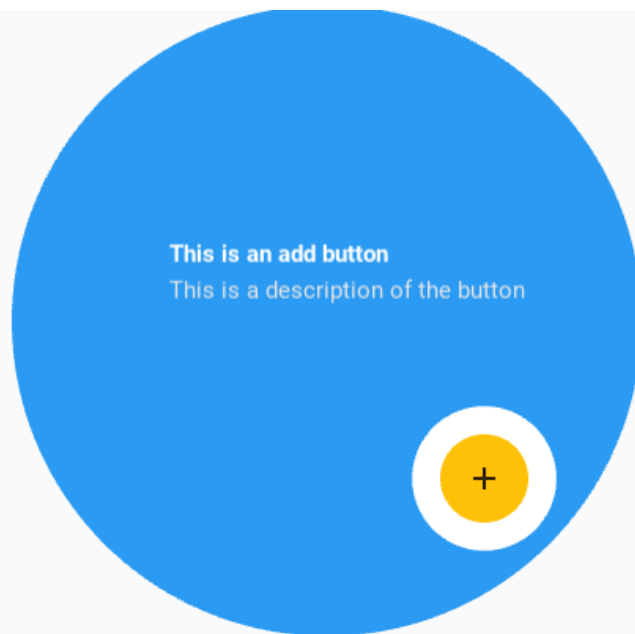

```
self.tap_target_view = MDTapTargetView(  
    ...  
    widget_position="right_top",  
)
```



```
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(  
    ...  
    title_position="center",  
    widget_position="center",  
)
```

(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

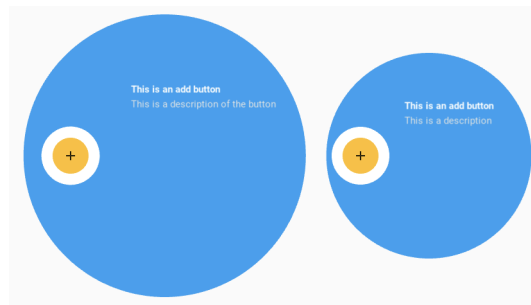
Called at the time of the start of the widget closed animation.

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`.

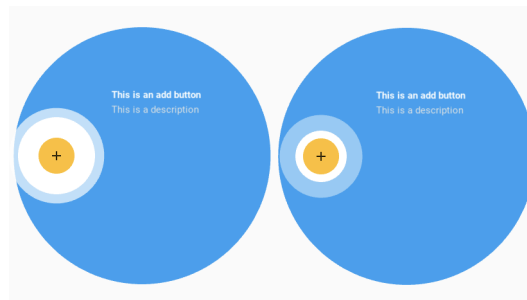
outer_circle_alpha

Alpha value for outer circle.

`outer_circle_alpha` is an `NumericProperty` and defaults to `0.96`.

target_radius

Radius for target circle.

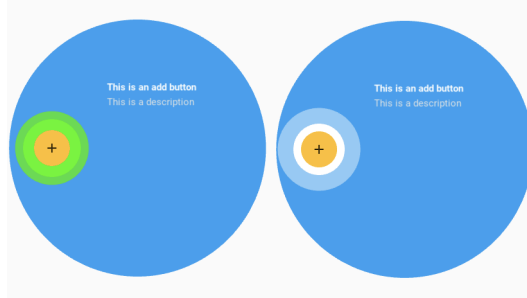


`target_radius` is an `NumericProperty` and defaults to `dp(45)`.

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, 1, 1]`.

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)
```

2.3.5 ScrollView

New in version 1.0.0.

[ScrollView](#) class equivalent. Simplifies working with some widget properties. For example:

ScrollView

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

MDScrollView

```
MDScrollView:
    md_bg_color: app.theme_cls.primary_color
```

API - `kivymd.uix.scrollview`

```
class kivymd.uix.scrollview.MDScrollView(*args, **kwargs)
```

ScrollView class. For more information, see in the [ScrollView](#) class documentation.

2.3.6 ResponsiveLayout

New in version 1.0.0.

Responsive design is a graphic user interface (GUI) design approach used to create content that adjusts smoothly to various screen sizes.

The *MDResponsiveLayout* class does not reorganize your UI. Its task is to track the size of the application screen and, depending on this size, the *MDResponsiveLayout* class selects which UI layout should be displayed at the moment: mobile, tablet or desktop. Therefore, if you want to have a responsive view some kind of layout in your application, you should have three KV files with UI markup for three platforms.

You need to set three parameters for the *MDResponsiveLayout* class *mobile_view*, *tablet_view* and *desktop_view*. These should be Kivy or KivyMD widgets.

Usage responsive

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.label import MDLabel
from kivymd.uix.responsivelayout import MDResponsiveLayout
from kivymd.uix.screen import MDScreen

KV = '''
<CommonComponentLabel>
    halign: "center"

<MobileView>
    CommonComponentLabel:
        text: "Mobile"

<TabletView>
    CommonComponentLabel:
        text: "Table"

<DesktopView>
    CommonComponentLabel:
        text: "Desktop"

ResponsiveView:
'''

class CommonComponentLabel(MDLabel):
    pass
```

(continues on next page)

(continued from previous page)

```
class MobileView(MDScreen):
    pass

class TabletView(MDScreen):
    pass

class DesktopView(MDScreen):
    pass

class ResponsiveView(MDResponsiveLayout, MDScreen):
    def __init__(self, **kw):
        super().__init__(**kw)
        self.mobile_view = MobileView()
        self.tablet_view = TabletView()
        self.desktop_view = DesktopView()

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

Test().run()
```

Note: Use common components for platform layouts (mobile, tablet, desktop views). As shown in the example above, such a common component is the *CommonComponentLabel* widget.

Perhaps you expected more from the *MDResponsiveLayout* widget, but even *Flutter* uses a similar approach to creating a responsive UI.

You can also use the *commands* provided to you by the developer tools to create a project with an responsive design.

API - `kivymd.uix.responsivelayout`

```
class kivymd.uix.responsivelayout.MDResponsiveLayout(*args, **kwargs)
```

Events

on_change_screen_type

Called when the screen type changes.

mobile_view

Mobile view. Must be a Kivy or KivyMD widget.

mobile_view is an *ObjectProperty* and defaults to *None*.

tablet_view

Tablet view. Must be a Kivy or KivyMD widget.

`tablet_view` is an `ObjectProperty` and defaults to `None`.

desktop_view

Desktop view. Must be a Kivy or KivyMD widget.

`desktop_view` is an `ObjectProperty` and defaults to `None`.

on_change_screen_type(self, *args)

Called when the screen type changes.

on_size(self, *args)

Called when the application screen size changes.

set_screen(self)

Sets the screen according to the type of application screen size: mobile/tablet or desktop view.

2.3.7 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 = '''
MDScreen:

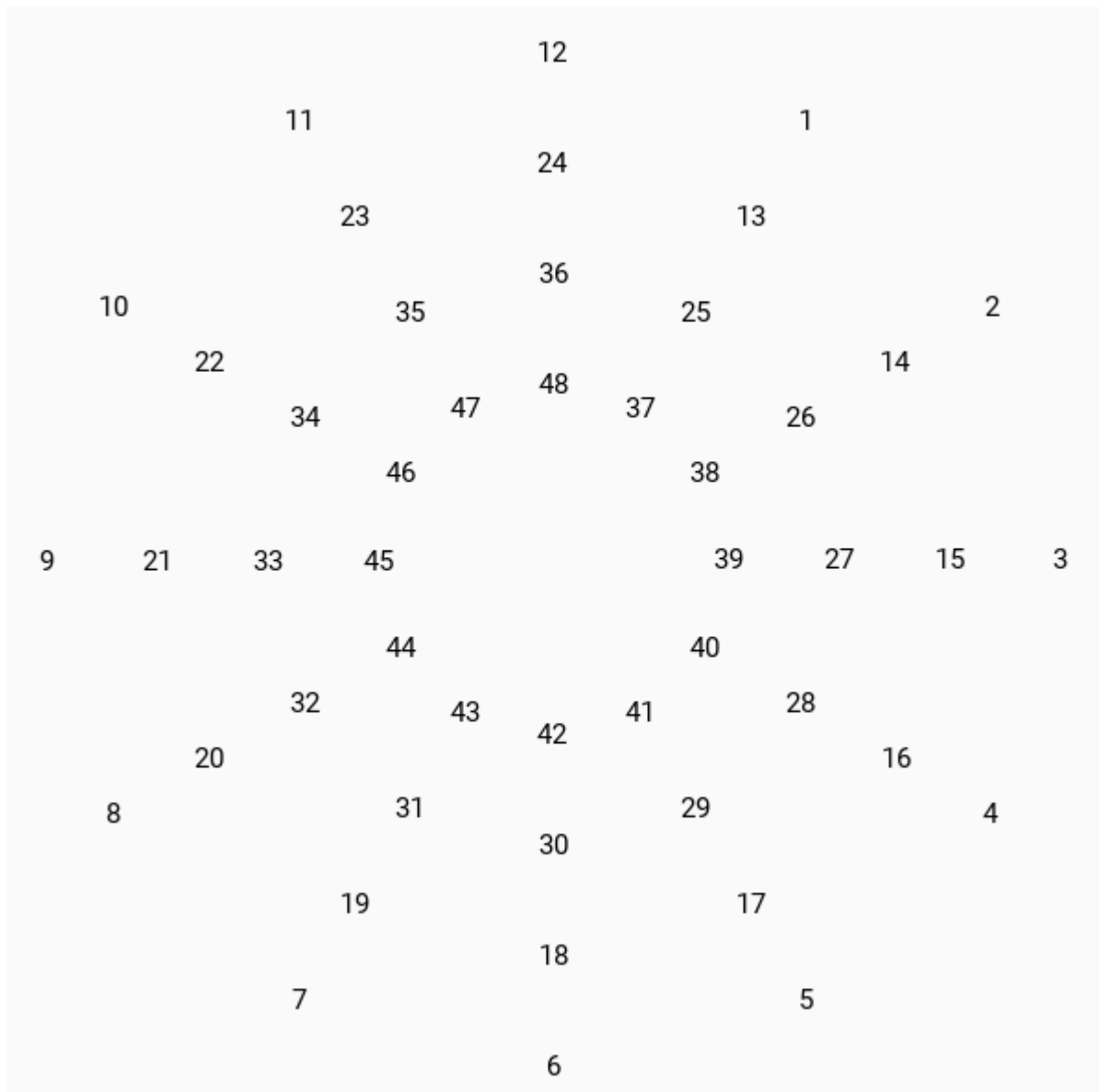
    MDCircularLayout:
        id: container
        pos_hint: {"center_x": .5, "center_y": .5}
        row_spacing: min(self.size) * 0.1
'''

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])
            )
```

(continues on next page)

(continued from previous page)

`Main().run()`

API - kivymd.uix.circularlayout

class kivymd.uix.circularlayout.MDCircularLayout(**kwargs)

Float layout class. For more information, see in the [FloatLayout](#) class documentation.

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 position 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*.

get_angle(self, pos: tuple)

Returns the angle of given pos.

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)
```

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.

2.3.8 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, 0]
```

MDScreen

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

API - `kivymd.uix.screen`

class `kivymd.uix.screen.MDScreen`(*args, **kwargs)

`Screen` is an element intended to be used with a `MDScreenManager`. For more information, see in the `Screen` class documentation.

hero_to

Must be a `MDHeroTo` class. See the documentation of the `MDHeroTo` widget for more detailed information.

Changed in version 1.0.0.

`hero_to` is an `ObjectProperty` and defaults to `None`.

on_hero_to(*self*, *screen*, *widget*)

2.3.9 ScreenManager

New in version 1.0.0.

`ScreenManager` class equivalent. If you want to use Hero animations you need to use `MDScreenManager` not `ScreenManager` class.

API - `kivymd.uix.screenmanager`

class `kivymd.uix.screenmanager.MDScreenManager(*args, **kwargs)`

Screen manager. This is the main class that will control your `MDScreen` stack and memory. For more information, see in the `ScreenManager` class documentation.

current_hero

The name of the current tag for the `MDHeroFrom` and `MDHeroTo` objects that will be animated when animating the transition between screens.

See the `Hero` module documentation for more information about creating and using Hero animations.

`current_hero` is an `StringProperty` and defaults to `None`.

check_transition(self, *args)

Sets the default type transition.

get_hero_from_widget(self)

Get an `MDHeroTo` object with the `current_hero` tag.

add_widget(self, widget, *args, **kwargs)

Changed in version 2.1.0.

Renamed argument `screen` to `widget`.

2.3.10 BoxLayout

`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
```

MDBoxLayout

```
MDBoxLayout:  
    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.boxlayout

class kivymd.uix.boxlayout.**MDBoxLayout**(*args, **kwargs)

Box layout class. For more information, see in the [BoxLayout](#) class documentation.

2.3.11 RecycleView

New in version 1.0.0.

[RecycleView](#) class equivalent. Simplifies working with some widget properties. For example:

RecycleView

```
RecycleView:

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

MDRecycleView

```
MDRecycleView:
    md_bg_color: app.theme_cls.primary_color
```

API - kivymd.uix.recycleview

class kivymd.uix.recycleview.**MDRecycleView**(*args, **kwargs)

Recycle view class. For more information, see in the [RecycleView](#) class documentation.

2.3.12 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(*args, **kwargs)
```

Stack layout class. For more information, see in the [StackLayout](#) class documentation.

2.3.13 RelativeLayout

[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(*args, **kwargs)
```

Relative layout class. For more information, see in the [RelativeLayout](#) class documentation.

2.3.14 Hero

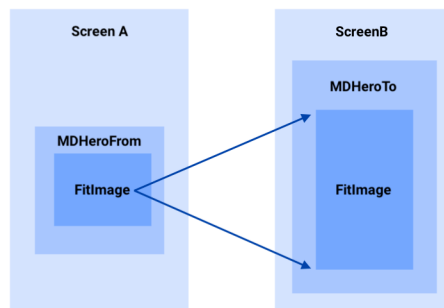
New in version 1.0.0.

Use the `MDHeroFrom` widget to animate a widget from one screen to the next.

- The hero refers to the widget that flies between screens.
- Create a hero animation using KivyMD's `MDHeroFrom` widget.
- Fly the hero from one screen to another.
- Animate the transformation of a hero's shape from circular to rectangular while flying it from one screen to another.
- The `MDHeroFrom` widget in KivyMD implements a style of animation commonly known as shared element transitions or shared element animations.

The widget that will move from screen A to screen B will be a hero. To move a widget from one screen to another using hero animation, you need to do the following:

- On screen **A**, place the `MDHeroFrom` container.
- Sets a tag (string) for the `MDHeroFrom` container.
- Place a hero in the `MDHeroFrom` container.
- On screen **B**, place the `MDHeroTo` container - our hero from screen **A** will fly into this container.



Warning: `MDHeroFrom` container cannot have more than one child widget.

Base example

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreenManager:
```

(continues on next page)

(continued from previous page)

```

MDScreen:
    name: "screen A"
    md_bg_color: "lightblue"

    MDHeroFrom:
        id: hero_from
        tag: "hero"
        size_hint: None, None
        size: "120dp", "120dp"
        pos_hint: {"top": .98}
        x: 24

        FitImage:
            source: "https://github.com/kivymd/internal/raw/main/logo/kivymd_logo_
↪blue.png"
            size_hint: None, None
            size: hero_from.size

    MDRaisedButton:
        text: "Move Hero To Screen B"
        pos_hint: {"center_x": .5}
        y: "36dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen B"

MDScreen:
    name: "screen B"
    hero_to: hero_to
    md_bg_color: "cadetblue"

    MDHeroTo:
        id: hero_to
        size_hint: None, None
        size: "220dp", "220dp"
        pos_hint: {"center_x": .5, "center_y": .5}

    MDRaisedButton:
        text: "Move Hero To Screen A"
        pos_hint: {"center_x": .5}
        y: "36dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen A"

'''

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

```

(continues on next page)

(continued from previous page)

```
Test().run()
```

Note that the child of the *MDHeroFrom* widget must have the size of the parent:

```
MDHeroFrom:
    id: hero_from

    FitImage:
        size_hint: None, None
        size: hero_from.size
```

To enable hero animation before setting the name of the current screen for the screen manager, you must specify the name of the tag of the *MDHeroFrom* container in which the hero is located:

```
MDRaisedButton:
    text: "Move Hero To Screen B"
    on_release:
        root.current_hero = "hero"
        root.current = "screen 2"
```

If you need to switch to a screen that does not contain heroes, set the *current_hero* attribute for the screen manager as "" (empty string):

```
MDRaisedButton:
    text: "Go To Another Screen"
    on_release:
        root.current_hero = ""
        root.current = "another screen"
```

Example

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreenManager:

    MDScreen:
        name: "screen A"
        md_bg_color: "lightblue"

        MDHeroFrom:
            id: hero_from
            tag: "hero"
            size_hint: None, None
            size: "120dp", "120dp"
            pos_hint: {"top": .98}
            x: 24
```

(continues on next page)

(continued from previous page)

```

        FitImage:
            source: "https://github.com/kivymd/internal/raw/main/logo/kivymd_logo_
↪blue.png"
            size_hint: None, None
            size: hero_from.size

    MDRaisedButton:
        text: "Move Hero To Screen B"
        pos_hint: {"center_x": .5}
        y: "36dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen B"

    MDScreen:
        name: "screen B"
        hero_to: hero_to
        md_bg_color: "cadetblue"

    MDHeroTo:
        id: hero_to
        size_hint: None, None
        size: "220dp", "220dp"
        pos_hint: {"center_x": .5, "center_y": .5}

    MDRaisedButton:
        text: "Go To Screen C"
        pos_hint: {"center_x": .5}
        y: "52dp"
        on_release:
            root.current_hero = ""
            root.current = "screen C"

    MDRaisedButton:
        text: "Move Hero To Screen A"
        pos_hint: {"center_x": .5}
        y: "8dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen A"

    MDScreen:
        name: "screen C"

    MDLabel:
        text: "Screen C"
        halign: "center"

    MDRaisedButton:
        text: "Back To Screen B"
        pos_hint: {"center_x": .5}

```

(continues on next page)

(continued from previous page)

```

        y: "36dp"
        on_release:
            root.current = "screen B"
'''

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

Test().run()

```

Events

Two events are available for the hero:

- `on_transform_in` - when the hero flies from screen **A** to screen **B**.
- `on_transform_out` - when the hero back from screen **B** to screen **A**.

The `on_transform_in`, `on_transform_out` events relate to the `MDHeroFrom` container. For example, let's change the radius and background color of the hero during the flight between the screens:

```

from kivy import utils
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.hero import MDHeroFrom
from kivymd.uix.relativelayout import MDRelativeLayout

KV = '''
MDScreenManager:

    MDScreen:
        name: "screen A"
        md_bg_color: "lightblue"

        MyHero:
            id: hero_from
            tag: "hero"
            size_hint: None, None
            size: "120dp", "120dp"
            pos_hint: {"top": .98}
            x: 24

            MDRelativeLayout:
                size_hint: None, None
                size: hero_from.size

```

(continues on next page)

(continued from previous page)

```

        md_bg_color: "blue"
        radius: [24, 12, 24, 12]

        FitImage:
            source: "https://github.com/kivymd/internal/raw/main/logo/kivymd_
↪logo_blue.png"

    MDRaisedButton:
        text: "Move Hero To Screen B"
        pos_hint: {"center_x": .5}
        y: "36dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen B"

    MDScreen:
        name: "screen B"
        hero_to: hero_to
        md_bg_color: "cadetblue"

    MDHeroTo:
        id: hero_to
        size_hint: None, None
        size: "220dp", "220dp"
        pos_hint: {"center_x": .5, "center_y": .5}

    MDRaisedButton:
        text: "Move Hero To Screen A"
        pos_hint: {"center_x": .5}
        y: "36dp"
        on_release:
            root.current_hero = "hero"
            root.current = "screen A"
'''

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

class MyHero(MDHeroFrom):
    def on_transform_in(
        self, instance_hero_widget: MDRelativeLayout, duration: float
    ):
        '''
        Called when the hero flies from screen **A** to screen **B**.

        :param instance_hero_widget: dchild widget of the `MDHeroFrom` class.
        :param duration: duration of the transition animation between screens.
        '''

```

(continues on next page)

(continued from previous page)

```

        Animation(
            radius=[12, 24, 12, 24],
            duration=duration,
            md_bg_color=(0, 1, 1, 1),
        ).start(instance_hero_widget)

    def on_transform_out(
        self, instance_hero_widget: MDRelativeLayout, duration: float
    ):
        '''Called when the hero back from screen **B** to screen **A**.'''

        Animation(
            radius=[24, 12, 24, 12],
            duration=duration,
            md_bg_color=get_color_from_hex(utils.hex_colormap["blue"]),
        ).start(instance_hero_widget)

Test().run()

```

Usage with ScrollView

```

from kivy.animation import Animation
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.properties import StringProperty, ObjectProperty

from kivymd.app import MDApp
from kivymd.uix.hero import MDHeroFrom

KV = '''
<HeroItem>
    size_hint_y: None
    height: "200dp"
    radius: 24

    MDSmartTile:
        id: tile
        radius: 24
        box_radius: 0, 0, 24, 24
        box_color: 0, 0, 0, .5
        source: "image.jpg"
        size_hint: None, None
        size: root.size
        mipmap: True
        on_release: root.on_release()

    MDLabel:
        text: root.tag

```

(continues on next page)

(continued from previous page)

```

        bold: True
        font_style: "H6"
        opposite_colors: True

MDScreenManager:

    MDScreen:
        name: "screen A"

        ScrollView:

            MDGridLayout:
                id: box
                cols: 2
                spacing: "12dp"
                padding: "12dp"
                adaptive_height: True

    MDScreen:
        name: "screen B"
        hero_to: hero_to

        MDHeroTo:
            id: hero_to
            size_hint: 1, None
            height: "220dp"
            pos_hint: {"top": 1}

        MDRaisedButton:
            text: "Move Hero To Screen A"
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_hero = "hero"
                root.current = "screen A"
'''

class HeroItem(MDHeroFrom):
    text = StringProperty()
    manager = ObjectProperty()

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.ids.tile.ids.image.ripple_duration_in_fast = 0.05

    def on_transform_in(self, instance_hero_widget, duration):
        Animation(
            radius=[0, 0, 0, 0],
            box_radius=[0, 0, 0, 0],
            duration=duration,

```

(continues on next page)

(continued from previous page)

```

        ).start(instance_hero_widget)

    def on_transform_out(self, instance_hero_widget, duration):
        Animation(
            radius=[24, 24, 24, 24],
            box_radius=[0, 0, 24, 24],
            duration=duration,
        ).start(instance_hero_widget)

    def on_release(self):
        def switch_screen(*args):
            self.manager.current_hero = self.tag
            self.manager.current = "screen B"

        Clock.schedule_once(switch_screen, 0.2)

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

    def on_start(self):
        for i in range(12):
            hero_item = HeroItem(
                text=f"Item {i + 1}", tag=f"Tag {i}", manager=self.root
            )
            if not i % 2:
                hero_item.md_bg_color = "lightgrey"
            self.root.ids.box.add_widget(hero_item)

Test().run()

```

API - kivymd.uix.hero

class kivymd.uix.hero.MDHeroFrom(**kwargs)

The container from which the hero begins his flight.

Events

on_transform_in

when the hero flies from screen **A** to screen **B**.

on_transform_out

Called when the hero back from screen **B** to screen **A**.

tag

Tag ID for heroes.

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

on_transform_in(*self*, *args)

Called when the hero flies from screen **A** to screen **B**.

on_transform_out(*self*, *args)

Called when the hero back from screen **B** to screen **A**.

class kivymd.uix.hero.**MDHeroTo**(*args, **kwargs)

The container in which the hero comes.

2.3.15 GridLayout

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**(*args, **kwargs)

Implements the creation and addition of child widgets as declarative programming style.

2.3.16 Carousel

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

Carousel

```
kv='''
    YourCarousel:
        BoxLayout:
            [...]
        BoxLayout:
            [...]
```

(continues on next page)

(continued from previous page)

```

        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):
        ["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(*args, **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. remember to bind and unbind to this method.

on_slide_complete(self, *args)

Event launched when the Slide animation is complete. remember to bind and unbind 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.

2.3.17 FloatLayout

[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**(*args, **kwargs)

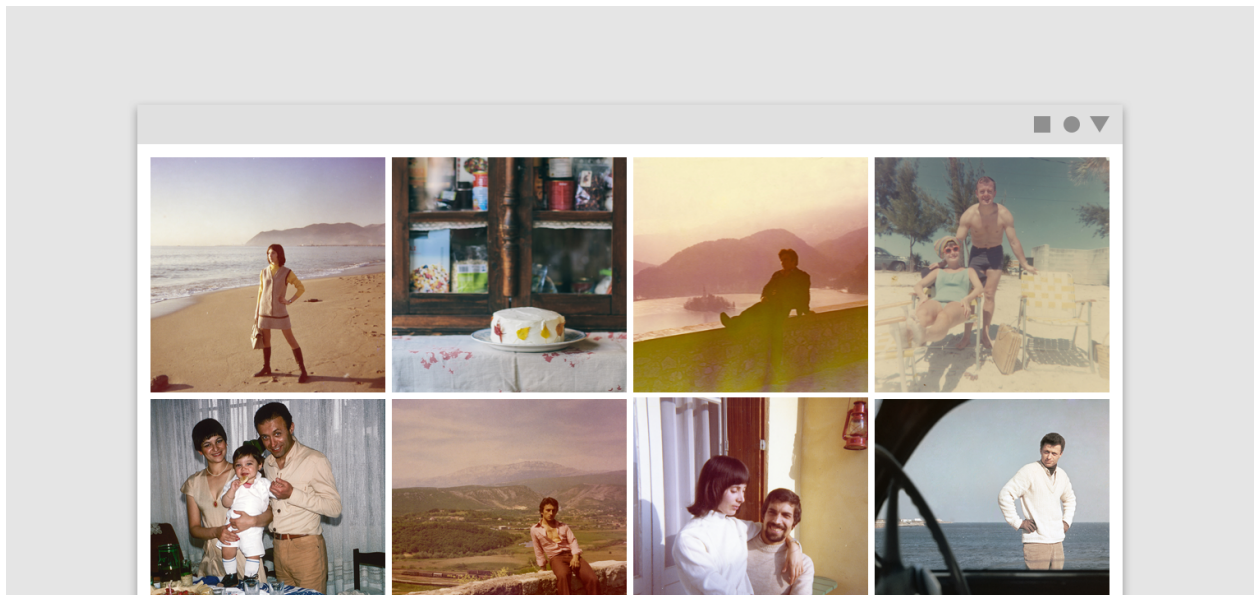
Float layout class. For more information, see in the [FloatLayout](#) class documentation.

2.3.18 ImageList

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:

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDSmartTile:
        radius: 24
        box_radius: [0, 0, 24, 24]
        box_color: 1, 1, 1, .2
        source: "cats.jpg"
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint: None, None
        size: "320dp", "320dp"

    MDIconButton:
        icon: "heart-outline"
        theme_icon_color: "Custom"
```

(continues on next page)

(continued from previous page)

```

        icon_color: 1, 0, 0, 1
        pos_hint: {"center_y": .5}
        on_release: self.icon = "heart" if self.icon == "heart-outline" else "heart-
        ↪outline"

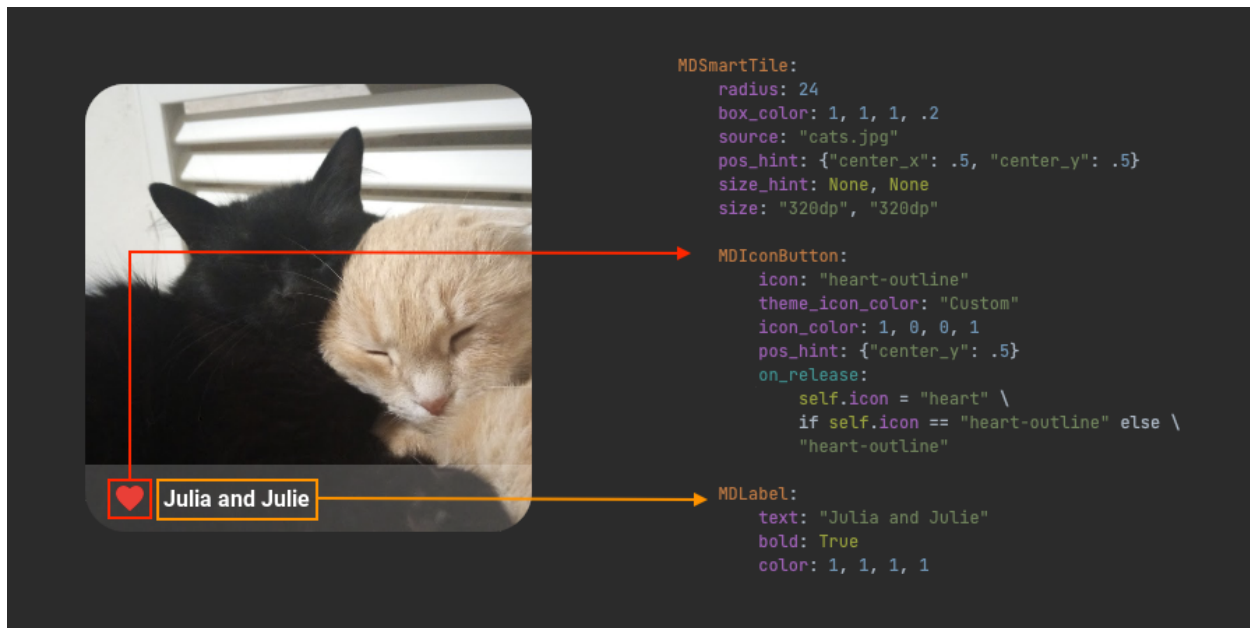
        MDLabel:
            text: "Julia and Julie"
            bold: True
            color: 1, 1, 1, 1
    ...

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

MyApp().run()

```

Implementation



API - `kivymd.uix.imagelist.imagelist`

2.3.19 RefreshLayout

Example

```
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.factory import Factory
from kivy.properties import StringProperty

from kivymd.app import MDApp
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@MDFloatLayout>

    MDBoxLayout:
        orientation: 'vertical'

        MDTopAppBar:
            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
''')
```

(continues on next page)

(continued from previous page)

```

class IconLeftSampleWidget(ILeftBodyTouch, MDIconButton):
    pass

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.refreshlayout`

class `kivymd.uix.refreshlayout.refreshlayout.MDScrollViewRefreshLayout(*args, **kwargs)`

ScrollView class. For more information, see in the [ScrollView](#) class documentation.

root_layout

The spinner will be attached to this layout.

`root_layout` is a [ObjectProperty](#) and defaults to *None*.

refresh_callback

The method that will be called at the `on_touch_up` event, provided that the overscroll of the list has been registered.

`refresh_callback` is a [ObjectProperty](#) and defaults to *None*.

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)**2.3.20 DataTables****See also:**

[Material Design spec, DataTables](#)

Data tables display sets of data across rows and columns.



<input type="checkbox"/>	Online	Astrid: NE shared ma
<input checked="" type="checkbox"/>	Offline	Cosmo: prod shared a
<input checked="" type="checkbox"/>	Online	Phoenix: prod shared l
<input type="checkbox"/>	Online	Sirius: prod shared an

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 2.1.0-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.datatables`

`class kivymd.uix.datatables.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 kivymd.app import MDApp
from kivymd.uix.datatables import MDDDataTable
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
        self.data_tables = MDDDataTable(
            use_pagination=True,
            check=True,
            column_data=[
                ("No.", dp(30)),
                ("Status", dp(30)),
                ("Signal Name", dp(60), self.sort_on_signal),
```

(continues on next page)

(continued from previous page)

```

        ("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",

```

(continues on next page)

(continued from previous page)

```

        (
            "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)
screen = MDScreen()
screen.add_widget(self.data_tables)
return screen

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 https://github.com/kivymd/KivyMD/pull/914 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 builtin 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]

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(
                [

```

(continues on next page)

(continued from previous page)

```

        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.

```

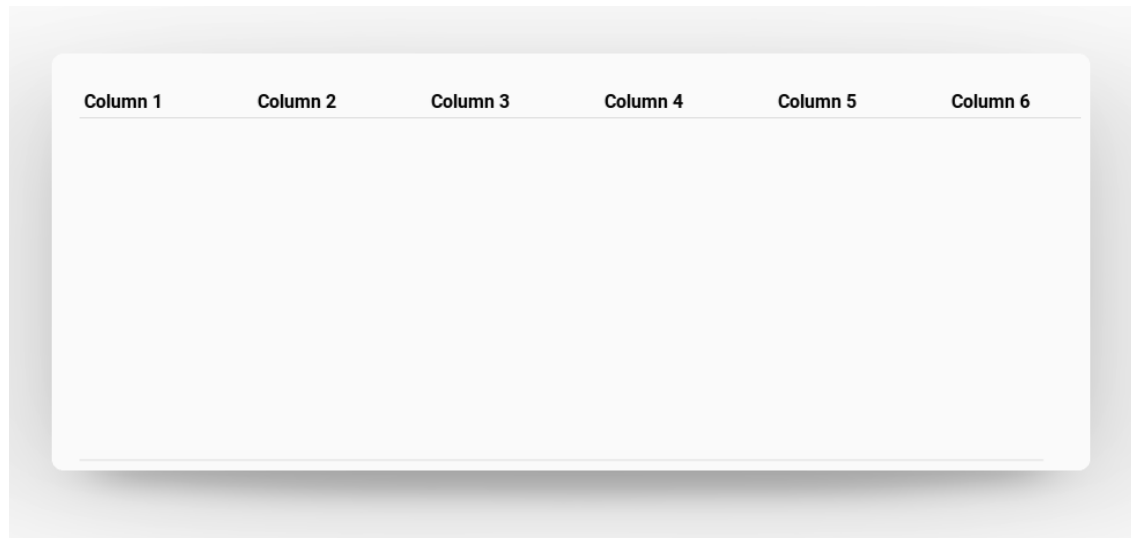
from kivy.metrics import dp

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

class Example(MDApp):
    def build(self):
        layout = AnchorLayout()
        self.data_tables = MDDataTable(
            size_hint=(0.7, 0.6),
            use_pagination=True,
            check=True,
            # name column, width column, sorting function column(optional)
            column_data=[
                ("No.", dp(30)),
                ("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_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",
    ],
    [
```

(continues on next page)

(continued from previous page)

```

        "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 following 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 MDDataTable

class Example(MDApp):
    def build(self):
        layout = AnchorLayout()
        data_tables = MDDataTable(
            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)),
            ]
        )
        layout.add_widget(data_tables)
        return layout

```

(continues on next page)

(continued from previous page)

```

        ("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",
            "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",
            (

```

(continues on next page)

(continued from previous page)

```






        "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]
        )
    )

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

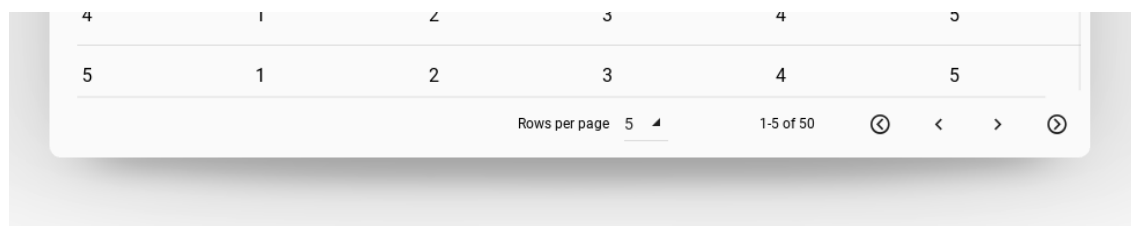
Use page pagination for table or not.

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

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

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.

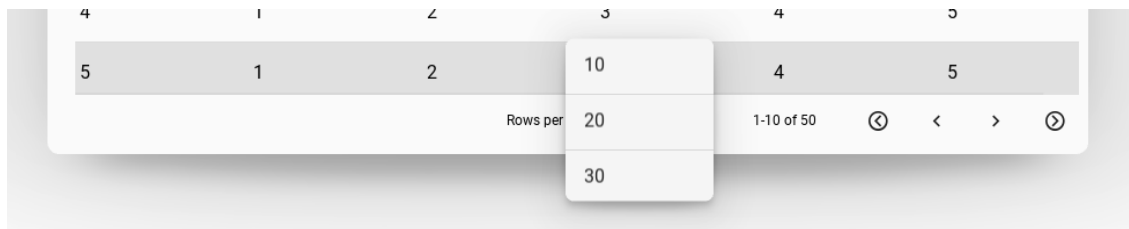
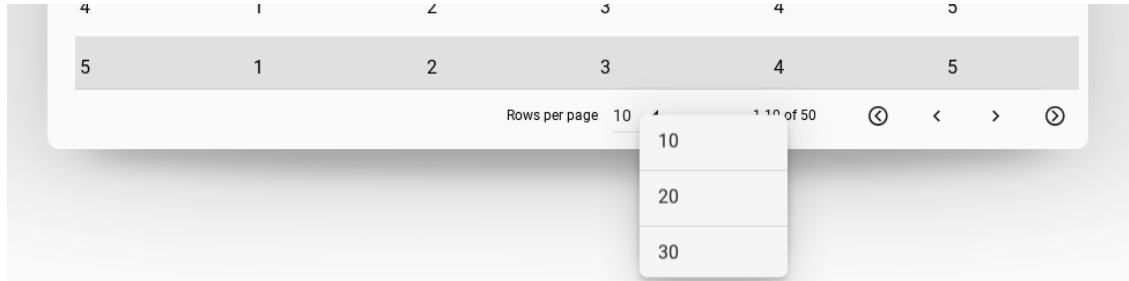
rows_num

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

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

pagination_menu_pos

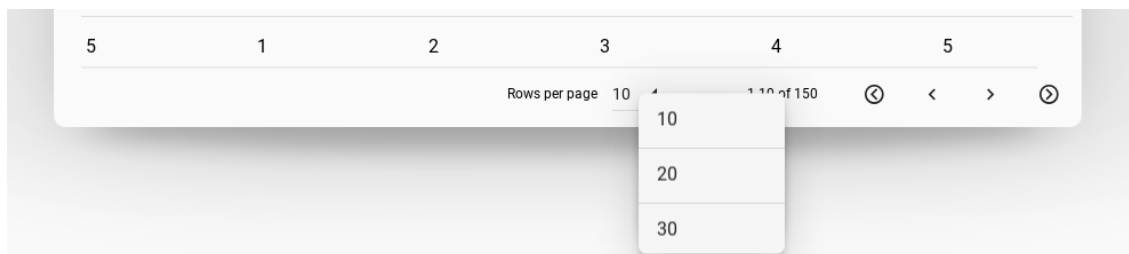
Menu position for selecting the number of displayed rows. Available options are `'center'`, `'auto'`.

Center**Auto**

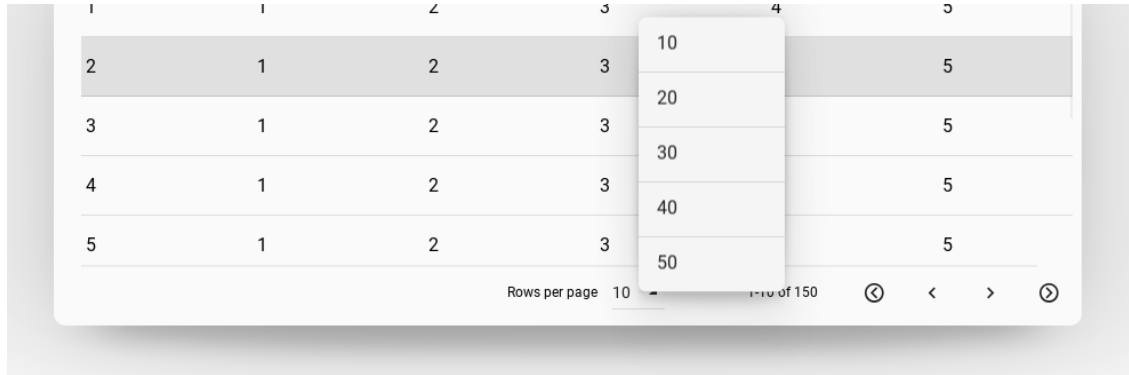
`pagination_menu_pos` is an `OptionProperty` and defaults to `'center'`.

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 MDDataTable

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)),
                ("[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)
```

(continues on next page)

(continued from previous page)

```
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]`.

background_color_header

Background color for `TableHeader` class.

New in version 1.0.0.

```
self.data_tables = MDDDataTable(
    ...,
    background_color_header="#65275d",
)
```

<input type="checkbox"/>	No.	Status	Signal Name	Severity	Stage
<input type="checkbox"/>	1	 No Signal	Astrid: NE shared managed	Medium	Triaged
<input type="checkbox"/>	2	 Offline	Cosmo: prod shared ares	Huge	Triaged
<input type="checkbox"/>	3	 Online	Phoenix: prod shared lyra-lists	Minor	Not Triaged

`background_color_header` is a `ColorProperty` and defaults to `None`.

background_color_cell

Background color for `CellRow` class.




New in version 1.0.0.

```
self.data_tables = MDDDataTable(
    ...,
    background_color_header="#65275d",
```

(continues on next page)

(continued from previous page)

```
background_color_cell="#451938",
)
```

<input type="checkbox"/>	No.	Status	Signal Name	Severity	Stage
<input type="checkbox"/>	1	 No Signal	Astrid: NE shared managed	Medium	Triaged
<input type="checkbox"/>	2	 Offline	Cosmo: prod shared ares	Huge	Triaged
<input type="checkbox"/>	3	 Online	Phoenix: prod shared lyra-lists	Minor	Not Triaged

`background_color_cell` is a `ColorProperty` and defaults to `None`.

background_color_selected_cell

Background selected color for `CellRow` class.

New in version 1.0.0.

```
self.data_tables = MDDataTable(
    ...,
    background_color_header="#65275d",
    background_color_cell="#451938",
    background_color_selected_cell="e4514f",
)
```

`background_color_selected_cell` is a `ColorProperty` and defaults to `None`.

effect_cls

Effect class. See `kivy/effects` package for more information.

New in version 1.0.0.

`effect_cls` is an `ObjectProperty` and defaults to `StiffScrollEffect`.

update_row_data(self, instance_data_table, data: list)

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.

add_row(self, data: Union[list, tuple])

Added new row to common table. Argument `data` is the row data from the list `row_data`.

Add/remove row

```

from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDataTable
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.button import MDRaisedButton

class Example(MDApp):
    data_tables = None

    def build(self):
        layout = MDFloatLayout()  # root layout
        # Creating control buttons.
        button_box = MDBoxLayout(
            pos_hint={"center_x": 0.5},
            adaptive_size=True,
            padding="24dp",
            spacing="24dp",
        )

        for button_text in ["Add row", "Remove row"]:
            button_box.add_widget(
                MDRaisedButton(
                    text=button_text, on_release=self.on_button_press
                )
            )

        # Create a table.
        self.data_tables = MDDataTable(
            pos_hint={"center_y": 0.5, "center_x": 0.5},
            size_hint=(0.9, 0.6),
            use_pagination=False,
            column_data=[
                ("No.", dp(30)),
                ("Column 1", dp(40)),
                ("Column 2", dp(40)),
                ("Column 3", dp(40)),
            ],
            row_data=[("1", "1", "2", "3")],
        )
        # Adding a table and buttons to the root layout.
        layout.add_widget(self.data_tables)
        layout.add_widget(button_box)

        return layout

    def on_button_press(self, instance_button: MDRaisedButton) -> None:
        '''Called when a control button is clicked.'''

```

(continues on next page)

(continued from previous page)

```

try:
    {
        "Add row": self.add_row,
        "Remove row": self.remove_row,
    }[instance_button.text]()
except KeyError:
    pass

def add_row(self) -> None:
    last_num_row = int(self.data_tables.row_data[-1][0])
    self.data_tables.add_row((str(last_num_row + 1), "1", "2", "3"))

def remove_row(self) -> None:
    if len(self.data_tables.row_data) > 1:
        self.data_tables.remove_row(self.data_tables.row_data[-1])

```

Example().run()

New in version 1.0.0.

remove_row(self, data: Union[list, tuple])

Removed row from common table. Argument *data* is the row data from the list *row_data*.

See the code in the doc string for the *add_row* method for more information.

New in version 1.0.0.

update_row(self, old_data: Union[list, tuple], new_data: Union[list, tuple])

Updates a table row. Argument *old_data/new_data* is the row data from the list *row_data*.

Update row

```

from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.ui.datatables import MDDataTable
from kivymd.ui.floatlayout import MDFloatLayout
from kivymd.ui.button import MDRaisedButton

class Example(MDApp):
    data_tables = None

    def build(self):
        layout = MDFloatLayout()
        layout.add_widget(
            MDRaisedButton(
                text="Change 2 row",
                pos_hint={"center_x": 0.5},
                on_release=self.update_row,
            )
        )

```

(continues on next page)

(continued from previous page)

```

        y=24,
    )
)
self.data_tables = MDDataTable(
    pos_hint={"center_y": 0.5, "center_x": 0.5},
    size_hint=(0.9, 0.6),
    use_pagination=False,
    column_data=[
        ("No.", dp(30)),
        ("Column 1", dp(40)),
        ("Column 2", dp(40)),
        ("Column 3", dp(40)),
    ],
    row_data=[(f"{i + 1}", "1", "2", "3") for i in range(3)],
)
layout.add_widget(self.data_tables)

return layout

def update_row(self, instance_button: MDRaisedButton) -> None:
    self.data_tables.update_row(
        self.data_tables.row_data[1], # old row data
        ["2", "A", "B", "C"],         # new row data
    )

```

Example().run()

New in version 1.0.0.

on_row_press(self, instance_cell_row)

Called when a table row is clicked.

on_check_press(self, row_data: list)

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

Parameters

row_data – One of the elements from the `MDDataTable.row_data` list.

get_row_checks(self)

Returns all rows that are checked.

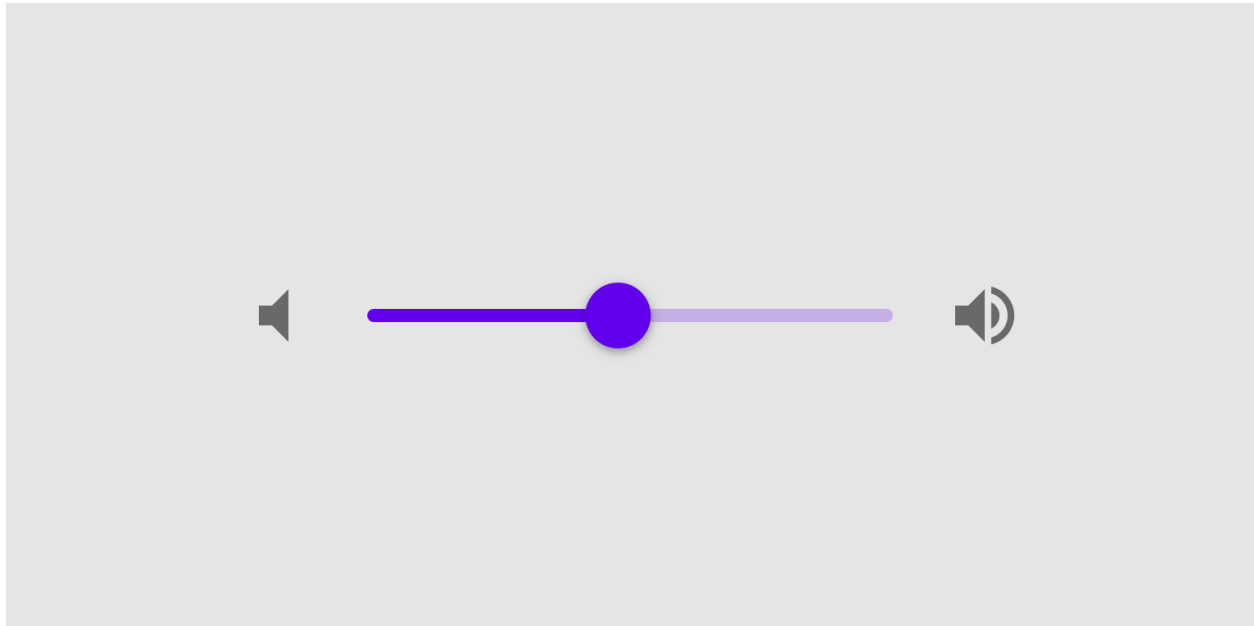
create_pagination_menu(self, interval: Union[int, float])

2.3.21 Slider

See also:

[Material Design spec, Sliders](#)

Sliders allow users to make selections from a range of values.



API - `kivymd.uix.slider.slider`

class `kivymd.uix.slider.slider.MDSlider(**kwargs)`

Class for creating a Slider widget. See in the [Slider](#) class documentation.

active

If the slider is clicked.

`active` is an [BooleanProperty](#) and defaults to *False*.

color

Color slider.

```
MDSlider
    color: "red"
```

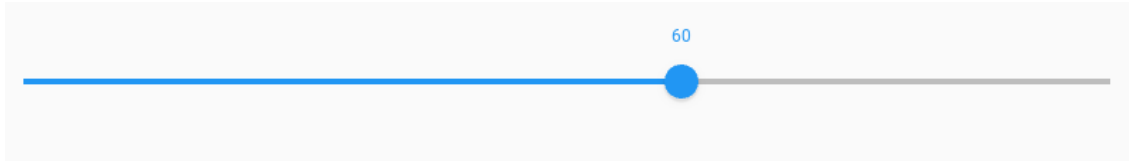


`color` is an [ColorProperty](#) and defaults to *None*.

hint

If True, then the current value is displayed above the slider.

```
MDSlider
    hint: True
```

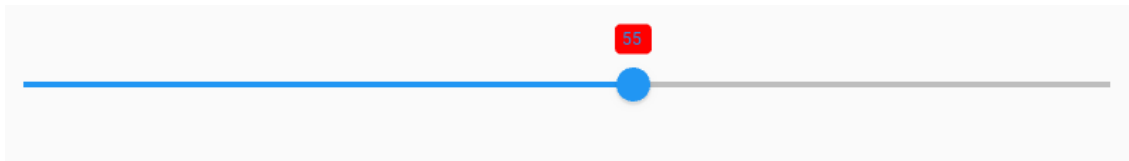


hint is an `BooleanProperty` and defaults to `True`.

hint_bg_color

Hint rectangle color in (r.g.b.a) format.

```
MDSlider
    hint: True
    hint_bg_color: "red"
```



hint_bg_color is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

hint_text_color

Hint text color in (r.g.b.a) format.

```
MDSlider
    hint: True
    hint_bg_color: "red"
    hint_text_color: "white"
```

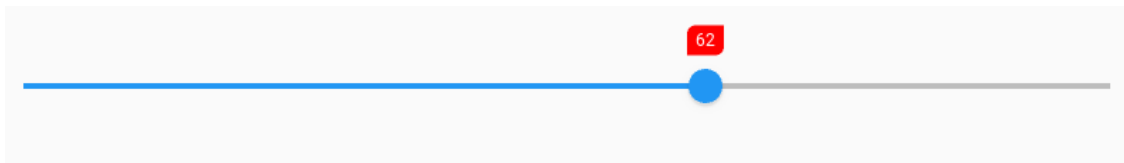


hint_text_color is an `ColorProperty` and defaults to `None`.

hint_radius

Hint radius.

```
MDSlider
    hint: True
    hint_bg_color: "red"
    hint_text_color: "white"
    hint_radius: [6, 0, 6, 0]
```



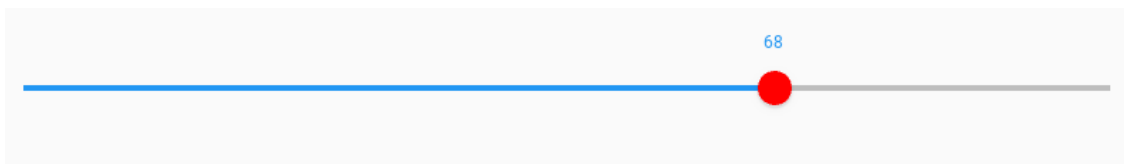
`hint_radius` is an `VariableListProperty` and defaults to `[dp(4), dp(4), dp(4), dp(4)]`.

thumb_color_active

The color of the thumb when the slider is active.

New in version 1.0.0.

```
MDSlider
    thumb_color_active: "red"
```



`thumb_color_active` is an `ColorProperty` and default to `None`.

thumb_color_inactive

The color of the thumb when the slider is inactive.

New in version 1.0.0.

```
MDSlider
    thumb_color_inactive: "red"
```



`thumb_color_inactive` is an `ColorProperty` and default to `None`.

thumb_color_disabled

The color of the thumb when the slider is in the disabled state.

New in version 1.0.0.

```
MDSlider
    value: 55
    disabled: True
    thumb_color_disabled: "red"
```



`thumb_color_disabled` is an `ColorProperty` and default to `None`.

track_color_active

The color of the track when the slider is active.

New in version 1.0.0.

```
MDSlider
    track_color_active: "red"
```



`track_color_active` is an `ColorProperty` and default to `None`.

track_color_inactive

The color of the track when the slider is inactive.

New in version 1.0.0.

```
MDSlider
    track_color_inactive: "red"
```



`track_color_inactive` is an `ColorProperty` and default to `None`.

track_color_disabled

The color of the track when the slider is in the disabled state.

New in version 1.0.0.

```
MDSlider
    disabled: True
    track_color_disabled: "red"
```



`track_color_disabled` is an `ColorProperty` and default to `None`.

show_off

Show the 'off' ring when set to minimum value.

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

`set_thumb_icon(self, *args)`

`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.

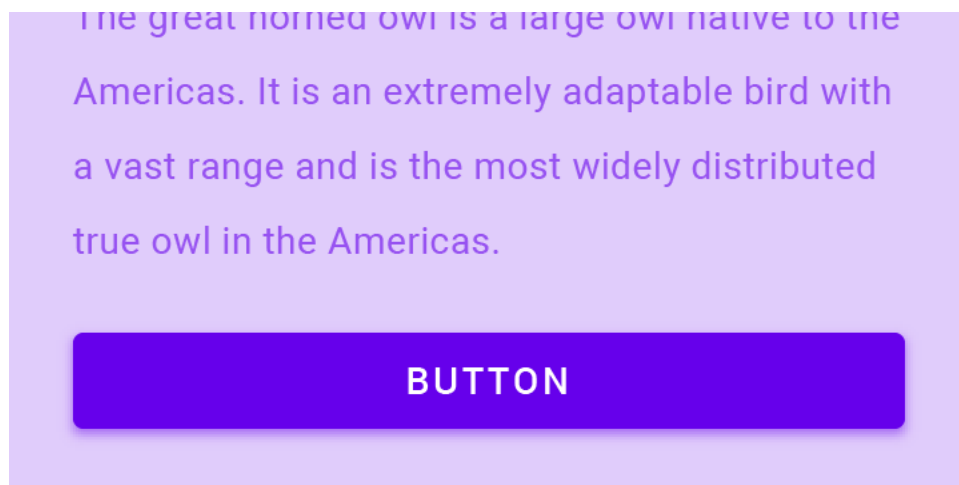
2.3.22 Button

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*
- *MDFillRoundFlatIconButton*
- *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}
'''

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:

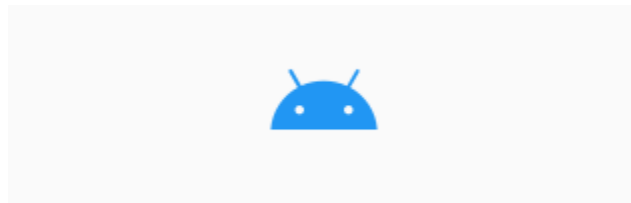
```
MDIconButton:
    icon: "data/logo/kivy-icon-256.png"
```

By default, *MDIconButton* button has a size (dp(48), dp (48)). Use *icon_size* attribute to resize the button:

```
MDIconButton:  
    icon: "android"  
    icon_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*, substituting *theme_icon_color* for *theme_text_color* and *icon_color* for *text_color*.

```
MDIconButton:  
    icon: "android"  
    theme_icon_color: "Custom"  
    icon_color: app.theme_cls.primary_color
```



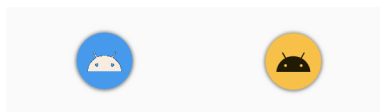
MDFloatingActionButton



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
```



Material design style 3

```

from kivy.lang import Builder

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

KV = '''
MDScreen:
    md_bg_color: "#f7f2fa"

    MDBoxLayout:
        id: box
        spacing: "56dp"
        adaptive_size: True
        pos_hint: {"center_x": .5, "center_y": .5}
'''

class TestNavigationDrawer(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(KV)

    def on_start(self):
        data = {
            "standard": {"md_bg_color": "#fefbff", "text_color": "#6851a5"},
            "small": {"md_bg_color": "#e9dff7", "text_color": "#211c29"},
            "large": {"md_bg_color": "#f8d7e3", "text_color": "#311021"},
        }
        for type_button in data.keys():
            self.root.ids.box.add_widget(
                MDFloatingActionButton(
                    icon="pencil",
                    type=type_button,
                    theme_icon_color="Custom",
                    md_bg_color=data[type_button]["md_bg_color"],
                    icon_color=data[type_button]["text_color"],
                )
            )

TestNavigationDrawer().run()

```

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
```



Or use markup:

```
MDFlatButton:
    text: "[color=#00ffcc]MDFLATBUTTON[/color]"
```

To specify the font size and font name, use the parameters as in the usual *Kivy* buttons:

```
MDFlatButton:
    text: "MDFLATBUTTON"
    font_size: "18sp"
    font_name: "path/to/font"
```

MDRaisedButton

This button is similar to the *MDFlatButton* 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
```



MDRectangleFlatIconButton



Button parameters *MDRectangleFlatIconButton* are the same as button *MDRectangleFlatButton*, with the addition of the *theme_icon_color* and *icon_color* parameters as for *MDIconButton*.

```
MDRectangleFlatIconButton:
    icon: "android"
    text: "MDRECTANGLEFLATICONBUTTON"
    theme_text_color: "Custom"
    text_color: 0, 0, 1, 1
    line_color: 1, 0, 1, 1
    theme_icon_color: "Custom"
    icon_color: 1, 0, 0, 1
```



Without border

```
from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
from kivymd.uix.button import MDRectangleFlatIconButton

class Example(MDApp):
    def build(self):
        screen = MDScreen()
        screen.add_widget(
            MDRectangleFlatIconButton(
                text="MDRectangleFlatIconButton",
                icon="language-python",
                line_color=(0, 0, 0, 0),
                pos_hint={"center_x": .5, "center_y": .5},
            )
        )
        return screen

Example().run()
```

```
MDRectangleFlatIconButton:
    text: "MDRectangleFlatIconButton"
```

(continues on next page)

(continued from previous page)

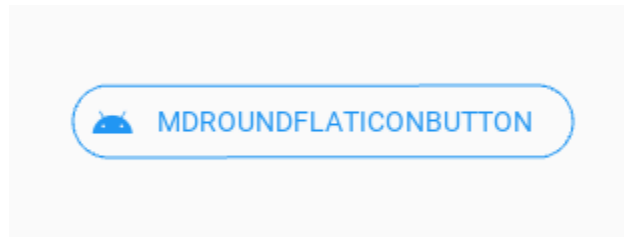
```
icon: "language-python"
line_color: 0, 0, 0, 0
pos_hint: {"center_x": .5, "center_y": .5}
```

MDRoundFlatButton

```
MDRoundFlatButton:
    text: "MDROUNDFLATBUTTON"
    text_color: 0, 1, 0, 1
```



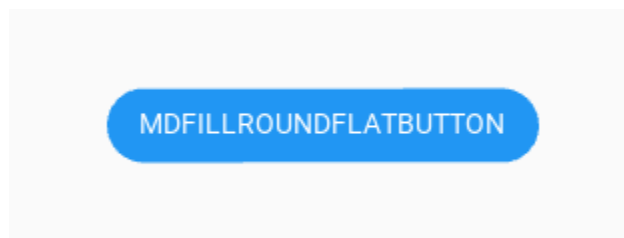
MDRoundFlatIconButton



Button parameters *MDRoundFlatIconButton* are the same as button *MDRoundFlatButton*, with the addition of the *theme_icon_color* and *icon_color* parameters as for *MDIconButton*:

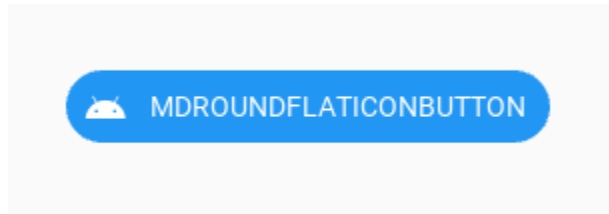
```
MDRoundFlatIconButton:
    icon: "android"
    text: "MDROUNDFLATICONBUTTON"
```

MDFillRoundFlatButton



Button parameters *MDFillRoundFlatButton* are the same as button *MDRaisedButton*.

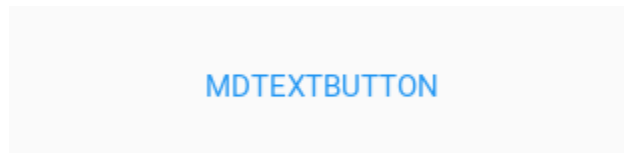
MDFillRoundFlatButton



Button parameters *MDFillRoundFlatButton* are the same as button *MDRaisedButton*, with the addition of the `theme_icon_color` and `icon_color` parameters as for *MDIconButton*.

Note: Notice that the width of the *MDFillRoundFlatButton* button matches the size of the button text.

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',
```

(continues on next page)

(continued from previous page)

```
        '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

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

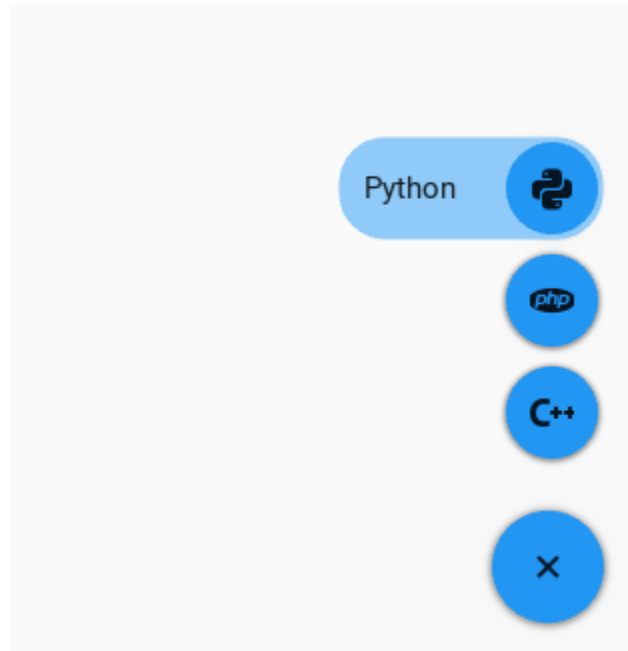
Example().run()
```

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.button`

class `kivymd.uix.button.button.BaseButton(**kwargs)`

Base class for all buttons.

padding

Padding between the widget box and its children, in pixels: [padding_left, padding_top, padding_right, padding_bottom].

padding also accepts a two argument form [padding_horizontal, padding_vertical] and a one argument form [padding].

New in version 1.0.0.

`padding` is a `VariableListProperty` and defaults to [16dp, 8dp, 16dp, 8dp].

halign

Horizontal anchor.

New in version 1.0.0.

`anchor_x` is an `OptionProperty` and defaults to 'center'. It accepts values of 'left', 'center' or 'right'.

valign

Vertical anchor.

New in version 1.0.0.

`anchor_y` is an `OptionProperty` and defaults to 'center'. It accepts values of 'top', 'center' or 'bottom'.

text

Button text.

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

icon

Button icon.

icon is a `StringProperty` and defaults to `''`.

font_style

Button text font style.

Available vanilla font_style are: `'H1'`, `'H2'`, `'H3'`, `'H4'`, `'H5'`, `'H6'`, `'Subtitle1'`, `'Subtitle2'`, `'Body1'`, `'Body2'`, `'Button'`, `'Caption'`, `'Overline'`, `'Icon'`.

font_style is a `StringProperty` and defaults to `'Body1'`.

theme_text_color

Button text type. Available options are: (`"Primary"`, `"Secondary"`, `"Hint"`, `"Error"`, `"Custom"`, `"ContrastParentBackground"`).

theme_text_color is an `OptionProperty` and defaults to `None` (set by button class).

theme_icon_color

Button icon type. Available options are: (`"Primary"`, `"Secondary"`, `"Hint"`, `"Error"`, `"Custom"`, `"ContrastParentBackground"`).

New in version 1.0.0.

theme_icon_color is an `OptionProperty` and defaults to `None` (set by button subclass).

text_color

Button text color in (r, g, b, a) format.

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

icon_color

Button icon color in (r, g, b, a) format.

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

font_name

Button text font name.

font_name is a `StringProperty` and defaults to `''`.

font_size

Button text font size.

font_size is a `NumericProperty` and defaults to `14sp`.

icon_size

Icon font size. Use this parameter as the font size, that is, in sp units.

New in version 1.0.0.

icon_size is a `NumericProperty` and defaults to `None`.

line_width

Line width for button border.

line_width is a `NumericProperty` and defaults to `1`.

line_color

Line color for button border.

line_color is a `ColorProperty` and defaults to `None`.

line_color_disabled

Disabled line color for button border.

New in version 1.0.0.

line_color_disabled is a [ColorProperty](#) and defaults to *None*.

md_bg_color

Button background color.

md_bg_color is a [ColorProperty](#) and defaults to *None*.

md_bg_color_disabled

The background color of the button when the button is disabled.

md_bg_color_disabled is a [ColorProperty](#) and defaults to *None*.

disabled_color

The color of the text and icon when the button is disabled, in the (r, g, b, a) format.

New in version 1.0.0.

disabled_color is a [ColorProperty](#) and defaults to *None*.

rounded_button

Should the button have fully rounded corners (e.g. like M3 buttons)?

New in version 1.0.0.

rounded_button is a [BooleanProperty](#) and defaults to *False*.

set_disabled_color(self, *args)

Sets the color for the icon, text and line of the button when button is disabled.

set_all_colors(self, *args)

Set all button colours.

set_button_colors(self, *args)

Set all button colours (except text/icons).

set_text_color(self, *args)

Set *_theme_text_color* and *_text_color* based on defaults and options.

set_icon_color(self, *args)

Set *_theme_icon_color* and *_icon_color* based on defaults and options.

set_radius(self, *args)

Set the radius, if we are a rounded button, based on the current height.

on_touch_down(self, touch)

Animates fade to background on press, for buttons with no background color.

on_touch_up(self, touch)

Animates return to original background on touch release.

on_disabled(self, instance_button, disabled_value: bool)**class kivymd.uix.button.button.MDFlatButton(**kwargs)**

A flat rectangular button with (by default) no border or background. Text is the default text color.

padding

Padding between the widget box and its children, in pixels: [padding_left, padding_top, padding_right, padding_bottom].

padding also accepts a two argument form [padding_horizontal, padding_vertical] and a one argument form [padding].

New in version 1.0.0.

padding is a [VariableListProperty](#) and defaults to [8dp, 8dp, 8dp, 8dp].

class kivymd.uix.button.button.MDRaisedButton(**kwargs)

A flat button with (by default) a primary color fill and matching color text.

class kivymd.uix.button.button.MDRectangleFlatButton(**kwargs)

A flat button with (by default) a primary color border and primary color text.

class kivymd.uix.button.button.MDRectangleFlatButton(**kwargs)

A flat button with (by default) a primary color border, primary color text and a primary color icon on the left.

class kivymd.uix.button.button.MDRoundFlatButton(**kwargs)

A flat button with (by default) fully rounded corners, a primary color border and primary color text.

class kivymd.uix.button.button.MDRoundFlatButton(**kwargs)

A flat button with (by default) rounded corners, a primary color border, primary color text and a primary color icon on the left.

class kivymd.uix.button.button.MDFillRoundFlatButton(**kwargs)

A flat button with (by default) rounded corners, a primary color fill and primary color text.

class kivymd.uix.button.button.MDFillRoundFlatButton(**kwargs)

A flat button with (by default) rounded corners, a primary color fill, primary color text and a primary color icon on the left.

class kivymd.uix.button.button.MDIconButton(**kwargs)

A simple rounded icon button.

icon

Button icon.

icon is a [StringProperty](#) and defaults to 'checkbox-blank-circle'.

set_size(self, interval: Union[int, float])

Sets the icon width/height based on the current *icon_size* attribute, or the default value if it is zero. The icon size is set to (48, 48) for an icon with the default font_size 24sp.

class kivymd.uix.button.button.MDFloatingActionButton(**kwargs)

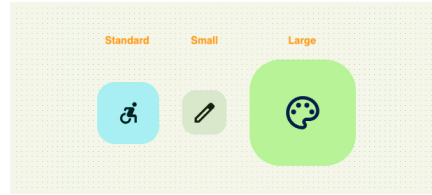
Implementation [FAB](#) button.

type

Type of M3 button.

New in version 1.0.0.

Available options are: 'small', 'large', 'standard'.



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

`set_font_size(self, *args)`

`set__radius(self, *args)`

`set_size(self, *args)`

`on_type(self, instance_md_floating_action_button, type: str)`

`class kivymd.uix.button.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 a `ColorProperty` and defaults to `None`.

`color_disabled`

Button color disabled in (r, g, b, a) format.

`color_disabled` is a `ColorProperty` and defaults to `None`.

`animation_label(self)`

`on_press(self, *args)`

`on_disabled(self, instance_button, disabled_value)`

This function hides the shadow when the widget is disabled. It sets the shadow to 0.

`class kivymd.uix.button.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_button: *MDFloatingBottomButton*)

Called when the mouse cursor goes outside the button of stack.

on_enter(*self*, instance_button: *MDFloatingBottomButton*)

Called when the mouse cursor is over a button from the stack.

on_data(*self*, instance_speed_dial, data: *dict*)

Creates a stack of buttons.

on_icon(*self*, instance_speed_dial, name_icon: *str*)**on_label_text_color**(*self*, instance_speed_dial, color: *list*)**on_color_icon_stack_button**(*self*, instance_speed_dial, color: *list*)**on_hint_animation**(*self*, instance_speed_dial, value: *bool*)**on_bg_hint_color**(*self*, instance_speed_dial, color: *list*)**on_color_icon_root_button**(*self*, instance_speed_dial, color: *list*)**on_bg_color_stack_button**(*self*, instance_speed_dial, color: *list*)**on_bg_color_root_button**(*self*, instance_speed_dial, color: *list*)**set_pos_labels**(*self*, instance_floating_label: *MDFloatingLabel*)

Sets the position of the floating labels. Called when the application's root window is resized.

set_pos_root_button(*self*, instance_floating_root_button: *MDFloatingRootButton*)

Sets the position of the root button. Called when the application's root window is resized.

set_pos_bottom_buttons(*self*, instance_floating_bottom_button: *MDFloatingBottomButton*)

Sets the position of the bottom buttons in a stack. Called when the application's root window is resized.

open_stack(*self*, instance_floating_root_button: *MDFloatingRootButton*)

Opens a button stack.

do_animation_open_stack(*self*, anim_data: *dict*)**Parameters**

anim_data –

{

```

    <kivymd.uix.button.MDFloatingBottomButton object>:
        <kivy.animation.Animation>,

    <kivymd.uix.button.MDFloatingBottomButton object>:
        <kivy.animation.Animation object>,

    ...,

}

close_stack(self)
    Closes the button stack.

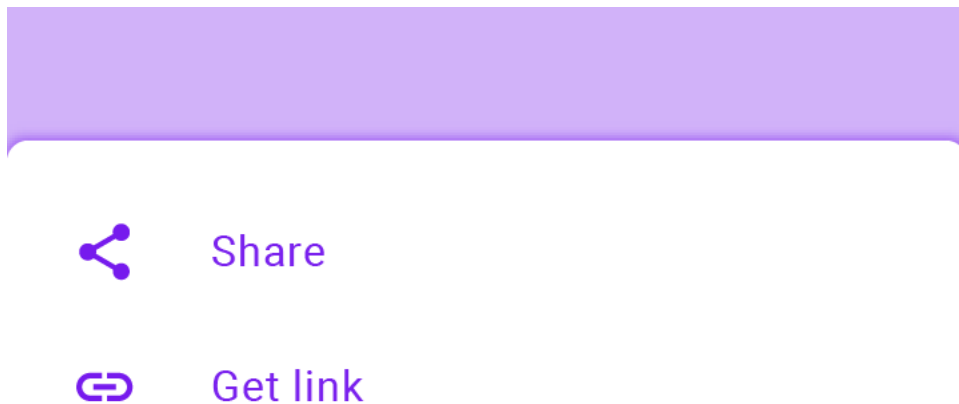
```

2.3.23 BottomSheet

See also:

Material Design spec, Sheets: bottom

Bottom sheets are surfaces containing supplementary content that are anchored to the bottom of the screen.



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 = '''
MDScreen:

    MDTopAppBar:
        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}",

```

(continues on next page)

(continued from previous page)

```

        lambda x, y=i: self.callback_for_menu_items(
            f"Standart Item {y}"
        ),
    )
    bottom_sheet_menu.open()

```

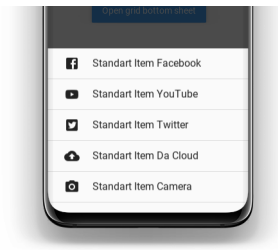
```
Example().run()
```

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 = '''
MDScreen:

    MDTopAppBar:
        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)

```

(continues on next page)

(continued from previous page)

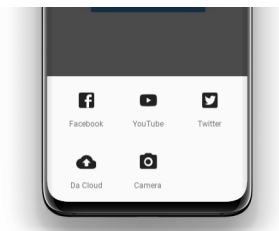
```

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),
            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>:

```

(continues on next page)

(continued from previous page)

```

orientation: "vertical"
size_hint_y: None
height: "400dp"

MDTopAppBar:
    title: 'Custom bottom sheet:'

ScrollView:

    MDGridLayout:
        cols: 1
        adaptive_height: True

        ItemForCustomBottomSheet:
            icon: "page-previous"
            text: "Preview"

        ItemForCustomBottomSheet:
            icon: "exit-to-app"
            text: "Exit"

MDScreen:

    MDTopAppBar:
        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}
...

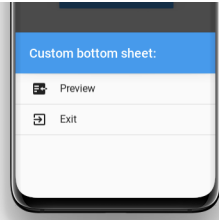
class Example(MDApp):
    custom_sheet = None

    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.bottomsheet`

class `kivymd.uix.bottomsheet.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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

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`.

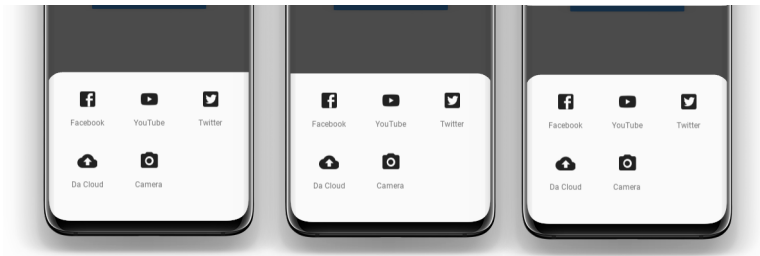
radius

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)

Display the modal in the 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* keyword 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 this animation, use:

```
view.dismiss(animation=False)
```

resize_content_layout(self, content, layout, interval=0)

class kivymd.uix.bottomsheet.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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

screen

Custom content.

screen is an `ObjectProperty` and defaults to *None*.

class kivymd.uix.bottomsheet.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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

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.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.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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

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;

2.3.24 SliverAppBar

New in version 1.0.0.

MDSliverAppBar is a Material Design widget in KivyMD which gives scrollable or collapsible MD-TopAppBar

Note: This widget is a modification of the [silverappbar.py](#) module.

Usage

MDScreen:

MDSliverAppBar:

MDSliverAppBarHeader:

Custom content.

...

Custom list.

MDSliverAppBarContent:



Title text

Subtitle text



Title text

Subtitle text

MDSliverAppBarContent



Title text

Subtitle text

Example

```
from kivy.lang.builder import Builder

from kivymd.uix.card import MDCard
from kivymd.uix.behaviors import RoundedRectangularElevationBehavior

KV = '''
<CardItem>
    size_hint_y: None
    height: "86dp"
    padding: "4dp"
    radius: 12
    elevation: 4

    FitImage:
        source: "avatar.jpg"
        radius: root.radius
        size_hint_x: None
        width: root.height

    MDBoxLayout:
        orientation: "vertical"
        adaptive_height: True
        spacing: "6dp"
        padding: "12dp", 0, 0, 0
        pos_hint: {"center_y": .5}

        MDLabel:
            text: "Title text"
            font_style: "H5"
            bold: True
            adaptive_height: True

        MDLabel:
            text: "Subtitle text"
            theme_text_color: "Hint"
            adaptive_height: True

MDScreen:

    MDSliverAppBar:
        background_color: "2d4a50"

        MDSliverAppBarHeader:

            MDRelativeLayout:

                FitImage:
                    source: "bg.jpg"
```

(continues on next page)

(continued from previous page)

```

        MDsliverAppBarContent:
            id: content
            orientation: "vertical"
            padding: "12dp"
            spacing: "12dp"
            adaptive_height: True
    ...

class CardItem(MDCard, RoundedRectangularElevationBehavior):
    pass

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

    def on_start(self):
        for x in range(10):
            self.root.ids.content.add_widget(CardItem())

Example().run()

```

API - `kivymd.uix.sliverappbar.sliverappbar`

class `kivymd.uix.sliverappbar.sliverappbar.MDsliverAppBarContent(**kwargs)`

Implements a box for a scrollable list of custom items.

md_bg_color

See [background_color](#).

md_bg_color is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

set_bg_color(self, interval: Union[int, float])

class `kivymd.uix.sliverappbar.sliverappbar.MDsliverAppBarHeader(*args, **kwargs)`

Box layout class. For more information, see in the [BoxLayout](#) class documentation.

class `kivymd.uix.sliverappbar.sliverappbar.MDsliverAppBar(*args, **kwargs)`

MDsliverAppBar class. See module documentation for more information.

Events

on_scroll_content

Called when the list of custom content is being scrolled.

toolbar_cls

Must be an object of the `MDTopAppBar` class documentation for more information.

By default, MDsliverAppBar widget uses the MDTopAppBar class with no parameters.

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

(continues on next page)

(continued from previous page)

```

from kivymd.uix.card import MDCard
from kivymd.uix.toolbar import MDTopAppBar
from kivymd.uix.behaviors import RoundedRectangularElevationBehavior

KV = '''
#:import SliverToolbar __main__.SliverToolbar

<CardItem>
    size_hint_y: None
    height: "86dp"
    padding: "4dp"
    radius: 12
    elevation: 4

    FitImage:
        source: "avatar.jpg"
        radius: root.radius
        size_hint_x: None
        width: root.height

    MDBoxLayout:
        orientation: "vertical"
        adaptive_height: True
        spacing: "6dp"
        padding: "12dp", 0, 0, 0
        pos_hint: {"center_y": .5}

        MDLabel:
            text: "Title text"
            font_style: "H5"
            bold: True
            adaptive_height: True

        MDLabel:
            text: "Subtitle text"
            theme_text_color: "Hint"
            adaptive_height: True

MDScreen:

    MDSliverAppBar:
        background_color: "2d4a50"
        toolbar_cls: SliverToolbar()

    MDSliverAppBarHeader:

        MDRelativeLayout:

            FitImage:
                source: "bg.jpg"

```

(continues on next page)

(continued from previous page)

```

        MDSliverAppBarContent:
            id: content
            orientation: "vertical"
            padding: "12dp"
            spacing: "12dp"
            adaptive_height: True
    ...

class CardItem(MDCard, RoundedRectangularElevationBehavior):
    pass

class SliverToolbar(MDTopAppBar):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.type_height = "medium"
        self.headline_text = "Headline medium"
        self.left_action_items = [["arrow-left", lambda x: x]]
        self.right_action_items = [
            ["attachment", lambda x: x],
            ["calendar", lambda x: x],
            ["dots-vertical", lambda x: x],
        ]

class Example(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(KV)

    def on_start(self):
        for x in range(10):
            self.root.ids.content.add_widget(CardItem())

Example().run()

```

`toolbar_cls` is an `ObjectProperty` and defaults to `None`.

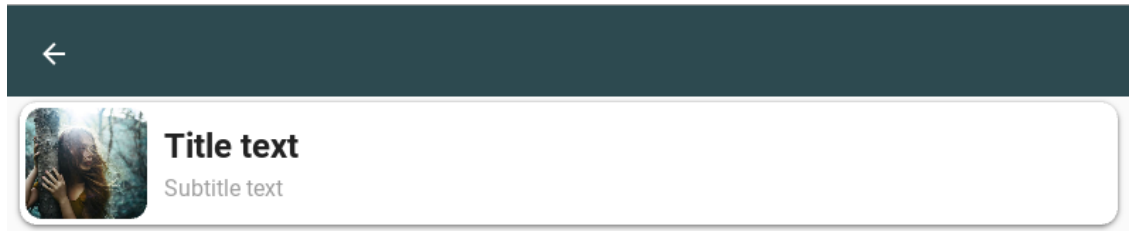
background_color

Background color of toolbar in (r, g, b, a) format.

```

MDSliverAppBar:
    background_color: "2d4a50"

```

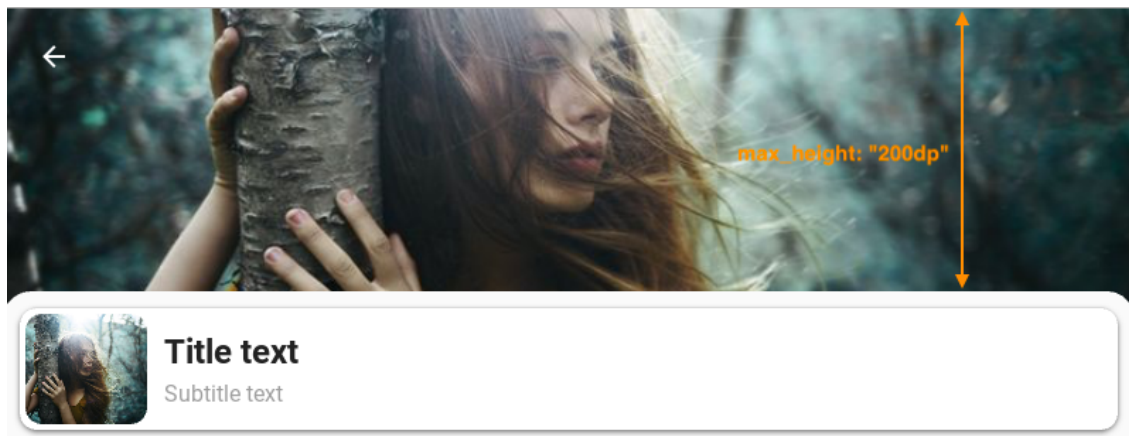


`background_color` is an `ColorProperty` and defaults to `None`.

max_height

Distance from top of screen to start of custom list content.

```
MDSliverAppBar:
    max_height: "200dp"
```



`max_height` is an `NumericProperty` and defaults to `Window.height / 2`.

hide_toolbar

Whether to hide the toolbar when scrolling through a list of custom content.

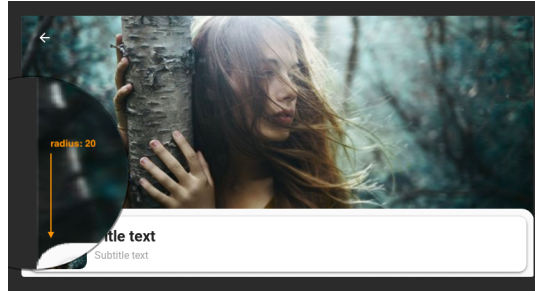
```
MDSliverAppBar:
    hide_toolbar: False
```

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

radius

Box radius for custom item list.

```
MDSliverAppBar:
    radius: 20
```



`radius` is an `VariableListProperty` and defaults to `[20]`.

max_opacity

Maximum background transparency value for the `MDSliverAppBarHeader` class.

```
MDSliverAppBar:
    max_opacity: .5
```

`max_opacity` is an `NumericProperty` and defaults to `1`.

on_scroll_content(*self*, *instance_sliverappbar*: *object* = *None*, *value*: *float* = *1.0*, *direction*: *str* = 'up')

Called when the list of custom content is being scrolled.

Parameters

- **instance_sliverappbar** – `MDSliverAppBar`
- **value** – see `scroll_y`
- **direction** – scroll direction: 'up/down'

on_background_color(*self*, *instance_sliver_appbar*, *color_value*: *list*)

on_toolbar_cls(*self*, *instance_sliver_appbar*, *instance_toolbar_cls*: `MDTopAppBar`)

Called when a value is set to the `toolbar_cls` parameter.

on_vbar(*self*)

get_default_toolbar(*self*)

Called if no value is passed for the `toolbar_cls` attribute.

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.

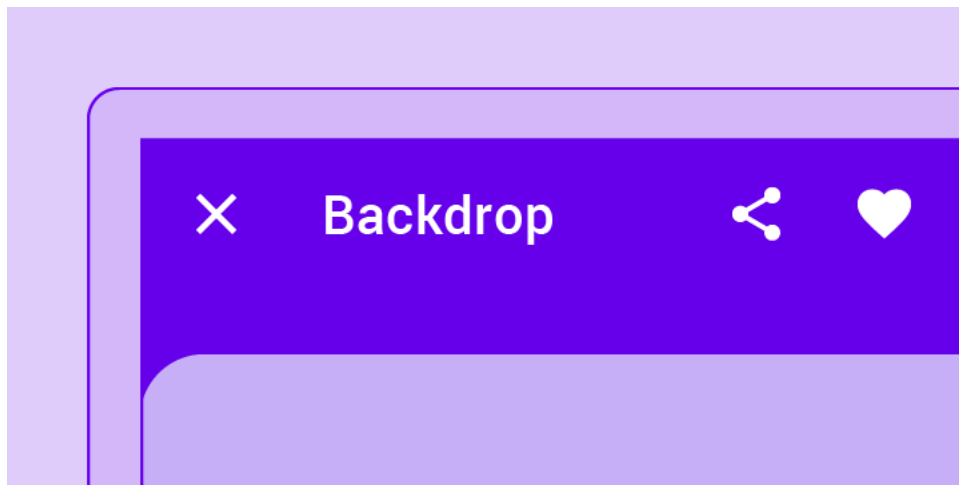
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.25 Backdrop

See also:

[Material Design spec, Backdrop](#)

Skeleton layout for using MDBackdrop:



Usage

[<Root>](#)

[MDBackdrop:](#)

[MDBackdropBackLayer:](#)

[ContentForBackdropBackLayer:](#)

[MDBackdropFrontLayer:](#)

(continues on next page)

(continued from previous page)

ContentForBackdropFrontLayer:**Example**

```

from kivy.lang import Builder

from kivymd.uix.screen import MDScreen
from kivymd.app import MDApp

# Your layouts.
Builder.load_string(
    '''
    #: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"

```

(continues on next page)

(continued from previous page)

```
radius_left: "25dp"
radius_right: "0dp"
header_text: "Menu:"

MDDropBackLayer:
    MyDropBackLayer:
        id: backlayer

MDDropFrontLayer:
    MyDropFrontLayer:
        backdrop: backdrop
'''
)

class ExampleBackdrop(MDScreen):
    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.backdrop`

```
class kivymd.uix.backdrop.backdrop.MDDropBack(*args, **kwargs)
```

Events

`on_open`

When the front layer drops.

`on_close`

When the front layer rises.

`anchor_title`

Position toolbar title. Only used with `material_style = 'M3'` Available options are: `'left'`, `'center'`, `'right'`.

New in version 1.0.0.

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

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.MDTopAppBar` in back layer. For more information, see the `kivymd.uix.toolbar.MDTopAppBar` 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.MDTopAppBar.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'`.

opening_time

The time taken for the panel to slide to the state `'open'`.

New in version 1.0.0.

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

opening_transition

The name of the animation transition type to use when animating to the state `'open'`.

New in version 1.0.0.

`opening_transition` is a `StringProperty` and defaults to `'out_quad'`.

closing_time

The time taken for the panel to slide to the state `'close'`.

New in version 1.0.0.

`closing_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'`.

New in version 1.0.0.

`closing_transition` is a `StringProperty` and defaults to `'out_quad'`.

on_open(self)

When the front layer drops.

on_close(self)

When the front layer rises.

on_left_action_items(self, instance_backdrop, menu: list)**on_header(self, instance_backdrop, value: bool)****open(self, open_up_to: int = 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.

animate_opacity_icon(self, instance_icon_menu: Union[ActionTopAppBarButton, None] = None, opacity_value: int = 0, call_set_new_icon: bool = True)

Starts the opacity animation of the icon.

set_new_icon(self, instance_animation: Animation, instance_icon_menu: ActionTopAppBarButton)

Sets the icon of the button depending on the state of the backdrop.

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.backdrop.MDBackdropToolbar(**kwargs)

Implements a toolbar for back content.

class kivymd.uix.backdrop.backdrop.MDBackdropFrontLayer(*args, **kwargs)

Container for front content.

class kivymd.uix.backdrop.backdrop.MDBackdropBackLayer(*args, **kwargs)

Container for back content.

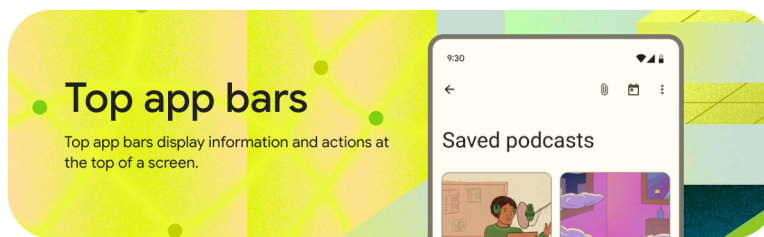
2.3.26 Toolbar

See also:

Material Design spec, App bars: top

Material Design spec, App bars: bottom

Material Design 3 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

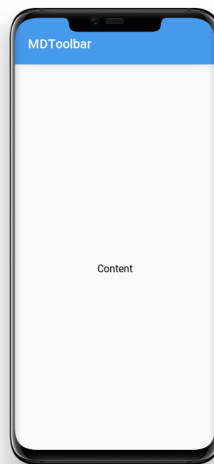
KV = '''
MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
        title: "MDTopAppBar"

    MDLabel:
        text: "Content"
        halign: "center"
'''

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

Test().run()
```



Add left menu

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


 A blue MDToolbar widget with the text "MDToolbar" on the left and a vertical ellipsis icon (three dots) on the right, representing a left menu.

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

Add right menu

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


 A blue MDToolbar widget with the text "MDToolbar" on the left and a vertical ellipsis icon (three dots) on the right, representing a right menu.

Add two item to the right menu

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


 A blue MDToolbar widget with the text "MDToolbar" on the left and two icons on the right: a vertical ellipsis icon (three dots) and a clock icon, representing two items in the right menu.

Change toolbar color

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



MDToolbar

Change toolbar text color

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



MDToolbar

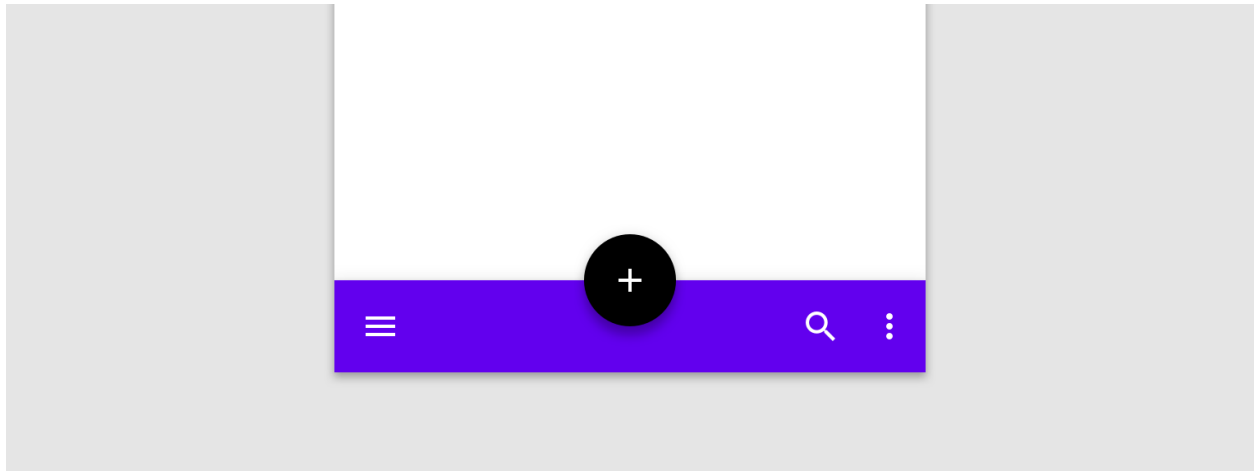
Shadow elevation control

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



MDToolbar

Bottom



Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDBoxLayout:

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

        MDTopAppBar:
            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:

    MDTopAppBar:
        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:

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




```

MDBottomAppBar:

    MDTopAppBar:
        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

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

```



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:

```

(continues on next page)

(continued from previous page)

```

orientation: "vertical"

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

```

Material design 3 style

```

from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.toolbar import MDTopAppBar

KV = '''
MDScreen:

    MDBoxLayout:
        id: box
        orientation: "vertical"
        spacing: "12dp"
        pos_hint: {"top": 1}
        adaptive_height: True
'''

class TestNavigationDrawer(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(KV)

    def on_start(self):
        for type_height in ["medium", "large", "small"]:
            self.root.ids.box.add_widget(

```

(continues on next page)

(continued from previous page)

```

MDTopAppBar(
    type_height=type_height,
    headline_text=f"Headline {type_height.lower()}",
    md_bg_color="#2d2734",
    left_action_items=[["arrow-left", lambda x: x]],
    right_action_items=[
        ["attachment", lambda x: x],
        ["calendar", lambda x: x],
        ["dots-vertical", lambda x: x],
    ],
    title="Title" if type_height == "small" else ""
)
)

```

```
TestNavigationDrawer().run()
```



API - kivymd.uix.toolbar.toolbar

```
class kivymd.uix.toolbar.toolbar.MDTopAppBar(**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:

```

MDTopAppBar:
    left_action_items: ["dots-vertical", callback, "tooltip text", "overflow_
↵text"]

```

icon_name - is a string that corresponds to an icon definition:

```

MDTopAppBar:
    right_action_items: [["home"]]

```



callback - is the function called on a touch release event and:

```
MDTopAppBar:
    right_action_items: [["home", lambda x: app.callback(x)]]
```

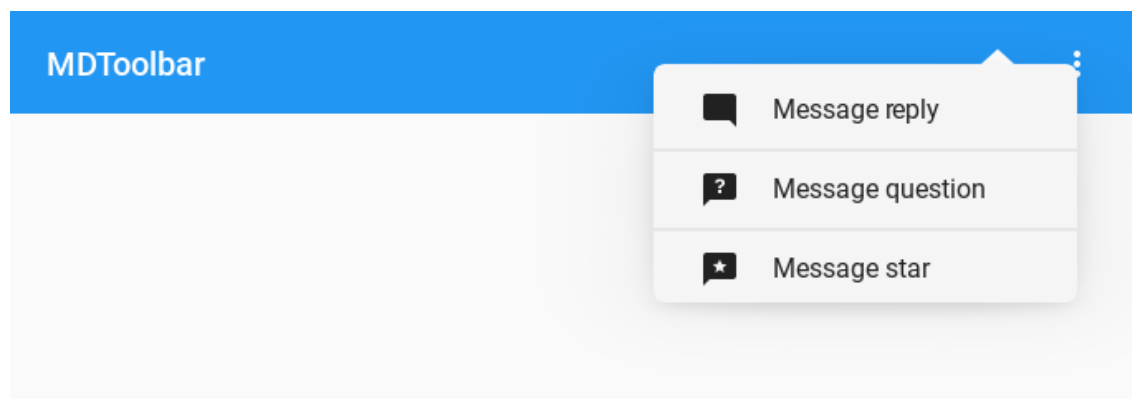
```
class Test(MDApp):
    def callback(self, instance_action_top_appbar_button):
        print(instance_action_top_appbar_button)
```

`tooltip text` - is the text to be displayed in the tooltip:

```
MDTopAppBar:
    right_action_items:
        [
            ["home", lambda x: app.callback(x), "Home"],
            ["message-star", lambda x: app.callback(x), "Message star"],
            ["message-question", lambda x: app.callback(x), "Message question"],
            ["message-reply", lambda x: app.callback(x), "Message reply"],
        ]
```

`overflow text` - is the text for menu items (OverflowMenuItem) of the corresponding action buttons:

```
MDTopAppBar:
    right_action_items:
        [
            ["home", lambda x: app.callback(x), "", "Home"],
            ["message-star", lambda x: app.callback(x), "", "Message star"],
            ["message-question", lambda x: app.callback(x), "", "Message question"],
            ["message-reply", lambda x: app.callback(x), "", "Message reply"],
        ]
```



Both the `callback` and `tooltip text` and `overflow 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 left of the toolbar. Works the same way as `left_action_items`.

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

title

Text toolbar.

```
MDTopAppBar:
    title: "MDTopAppBar"
```

MDToolbar

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

mode

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

Mode “end”:

```
MDBottomAppBar:
    MDTopAppBar:
        title: "Title"
        icon: "git"
        type: "bottom"
        left_action_items: [["menu", lambda x: x]]
        mode: "end"
```



Mode “free-end”:

```
MDBottomAppBar:
    MDTopAppBar:
        mode: "free-end"
```



Mode “free-center”:

```
MDBottomAppBar:

    MDTopAppBar:
        mode: "free-center"
```

**Mode “center”:**

```
MDBottomAppBar:

    MDTopAppBar:
        mode: "center"
```



mode is an *OptionProperty* and defaults to ‘center’.

type

When using the *MDBottomAppBar* class, the parameter *type* must be set to ‘bottom’:

```
MDBottomAppBar:

    MDTopAppBar:
        type: "bottom"
```

Available options are: ‘top’, ‘bottom’.

type is an *OptionProperty* and defaults to ‘top’.

opposite_colors

Changes the color of the label to the color opposite to the main theme.

```
MDTopAppBar:
    opposite_colors: True
```



```
MDTopAppBar:
    opposite_colors: True
```

MDToolbar

md_bg_bottom_color

The background color in (r, g, b, a) format for the toolbar with the bottom mode.

New in version 1.0.0.

```
MDBottomAppBar:

    MDTopAppBar:
        md_bg_bottom_color: 0, 1, 0, 1
        icon_color: self.md_bg_bottom_color
```



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

setBars_color

If `True` the background color of the bar status will be set automatically according to the current color of the toolbar.

New in version 1.0.0.

See `setBars_colors` <https://kivymd.readthedocs.io/en/latest/api/kivymd/utils/setBars_colors/> for more information.

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

use_overflow

As a top app bar is resized, actions move to the overflow menu from right to left.

New in version 1.0.0.

```
MDTopAppBar:
    title: "MDTopAppBar"
    use_overflow: True
    right_action_items:
        [
            ["home", lambda x: app.callback(x), "Home", "Home"],
            ["message-star", lambda x: app.callback(x), "Message star", "Message_↵star"],
            ["message-question", lambda x: app.callback(x), "Message question", ↵"Message question"],
            ["message-reply", lambda x: app.callback(x), "Message reply", "Message_↵reply"],
        ]
```

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

overflow_cls

Must be an object of the MDDropdownMenu class documentation for more information.

New in version 1.0.0.

```
from kivy.lang import Builder

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

KV = '''
#:import CustomOverFlowMenu __main__.CustomOverFlowMenu

MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
        title: "MDTopAppBar"
        use_overflow: True
        overflow_cls: CustomOverFlowMenu()
        right_action_items:
            [
                ["home", lambda x: app.callback(x), "Home", "Home"],
                ["message-star", lambda x: app.callback(x), "Message star",
↪ "Message star"],
                ["message-question", lambda x: app.callback(x), "Message question",
↪ "Message question"],
                ["message-reply", lambda x: app.callback(x), "Message reply",
↪ "Message reply"],
            ]

    MDLabel:
        text: "Content"
        halign: "center"
'''

class CustomOverFlowMenu(MDDropdownMenu):
    # In this class you can set custom properties for the overflow menu.
    pass

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

    def callback(self, instance_action_top_appbar_button):
        print(instance_action_top_appbar_button)

Test().run()
```

`overflow_cls` is an `ObjectProperty` and defaults to `None`.

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 *[]*.

anchor_title

Position toolbar title. Only used with *material_style = 'M3'* Available options are: *'left'*, *'center'*, *'right'*.

anchor_title is an *OptionProperty* and defaults to *None*.

headline_text

Headline text toolbar.

New in version 1.0.0.

headline_text is an *StringProperty* and defaults to *''*.

headline_text_color

Headline text color.

New in version 1.0.0.

headline_text_color is an *ColorProperty* and defaults to *None*.

type_height

Toolbar height type.

New in version 1.0.0.

Available options are: *'small'*, *'large'*, *'small'*.

type_height is an *OptionProperty* and defaults to *'small'*.

set_headline_font_style(*self*, *interval*: *Union[int, float]*)**on_width**(*self*, *instance_toolbar*, *width*: *float*)

Called when the toolbar is resized (size of the application window).

return_action_button_to_toolbar(*self*)**remove_overflow_button**(*self*)

Removes an overflow button to the toolbar.

add_overflow_button(*self*)

Adds an overflow button to the toolbar.

overflow_action_button_is_added(*self*)

Returns *True* if at least one action button (:class:`~ActionTopAppBarButton`) on the toolbar is added to the overflow.

add_action_button_to_overflow(*self*)

Adds an overflow button to the toolbar.

check_overflow_cls(*self*, *interval*: *Union[int, float]*)

If the user does not set the *overflow_cls* attribute but uses overflows, the *overflow_cls* attribute will use the default value.

on_type(*self*, *instance_toolbar*, *type_value*: *str*)
Called when the value of the **type** attribute changes.

on_type_height(*self*, *instance_toolbar*, *height_type_value*: *str*)
Called when the value of the **type_height** attribute changes.

on_action_button(*self*, **args*)
Method for the button used for the **MDBottomAppBar** class.

on_overflow_cls(*self*, *instance_toolbar*, *instance_overflow_cls*: **MDDropdownMenu**)
Called when the value of the **overflow_cls** attribute changes.

on_md_bg_color(*self*, *instance_toolbar*, *color_value*: *list*)
Called when the value of the **md_bg_color** attribute changes.

on_left_action_items(*self*, *instance_toolbar*, *items_value*: *list*)
Called when the value of the **left_action_items** attribute changes.

on_right_action_items(*self*, *instance_toolbar*, *items_value*: *list*)
Called when the value of the **right_action_items** attribute changes.

on_icon(*self*, *instance_toolbar*, *icon_name*: *str*)
Called when the value of the **icon** attribute changes.

on_icon_color(*self*, *instance*, *icon_name*: *str*)
Called when the value of the **icon_color** attribute changes.

on_md_bg_bottom_color(*self*, *instance_toolbar*, *color_value*: *list*)
Called when the value of the **md_bg_bottom_color** attribute changes.

on_anchor_title(*self*, *instance_toolbar*, *anchor_value*: *str*)
Called when the value of the **anchor_title** attribute changes.

on_mode(*self*, *instance_toolbar*, *mode_value*: *str*)
Called when the value of the **mode** attribute changes.

set_md_bg_color(*self*, *instance_toolbar*, *color_value*: *list*)

set_notch(*self*)

set_shadow(*self*, **args*)

get_default_overflow_cls(*self*)

update_overflow_menu_items(*self*, *action_button*)

update_bar_height(*self*, *instance_theme_manager*, *material_style_value*: *str*)

update_floating_radius(*self*, *interval*: *Union[int, float]*)

update_anchor_title(*self*, *material_style_value*: *str*)

update_action_bar(*self*, *instance_box_layout*, *action_bar_items*: *list*)

update_md_bg_color(*self*, **args*)

update_action_bar_text_colors(*self*, **args*)

remove_notch(*self*)

remove_shadow(*self*)

class kivymd.uix.toolbar.toolbar.**MDBottomAppBar**(*args, **kwargs)

Implements the creation and addition of child widgets as declarative programming style.

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.

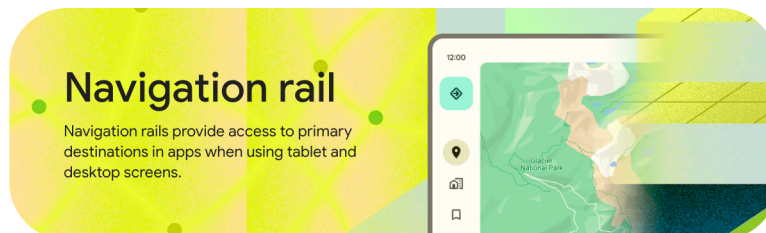
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.27 NavigationRail

New in version 1.0.0.

See also:

[Material Design spec](#), [Navigation rail](#)



Usage

```
MDNavigationRail:
```

```
    MDNavigationRailItem:
```

```
    MDNavigationRailItem:
```

```
    MDNavigationRailItem:
```

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

KV = '''

MDBoxLayout:

    MDNavigationRail:

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

        MDNavigationRailItem:
            text: "JavaScript"
            icon: "language-javascript"

        MDNavigationRailItem:
            text: "CPP"
            icon: "language-cpp"

        MDNavigationRailItem:
            text: "Git"
            icon: "git"

    MDScreen:
'''

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

Example().run()
```



Python



JavaScript



C++



Git

Example

```

from kivy.clock import Clock
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import RoundedRectangularElevationBehavior
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDFillRoundFlatIconButton
from kivymd.uix.label import MDLabel
from kivymd.uix.screen import MDScreen

KV = '''
#:import FadeTransition kivy.uix.screenmanager.FadeTransition

<ExtendedButton>
    elevation: 3
    -height: "56dp"

<DrawerClickableItem@MDNavigationDrawerItem>
    focus_color: "#e7e4c0"
    unfocus_color: "#fffcf4"

MDScreen:

    MDNavigationLayout:

        ScreenManager:

            MDScreen:

                MDBoxLayout:
                    orientation: "vertical"

                MDBoxLayout:
                    adaptive_height: True
                    md_bg_color: "#fffcf4"
                    padding: "12dp"

                    MDLabel:
                        text: "12:00"
                        adaptive_height: True
                        pos_hint: {"center_y": .5}

                MDBoxLayout:

                    MDNavigationRail:
                        id: navigation_rail
                        md_bg_color: "#fffcf4"
                        selected_color_background: "#e7e4c0"

```

(continues on next page)

(continued from previous page)

```

        ripple_color_item: "#e7e4c0"
        on_item_release: app.switch_screen(*args)

        MDNavigationRailMenuButton:
            on_release: nav_drawer.set_state("open")

        MDNavigationRailFabButton:
            md_bg_color: "#b0f0d6"

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

        MDNavigationRailItem:
            text: "JavaScript"
            icon: "language-javascript"

        MDNavigationRailItem:
            text: "CPP"
            icon: "language-cpp"

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

    ScreenManager:
        id: screen_manager
        transition:
            FadeTransition(duration=.2, clearcolor=app.theme_cls.bg_
↪dark)

    MDNavigationDrawer:
        id: nav_drawer
        radius: (0, 16, 16, 0)
        md_bg_color: "#fffcf4"
        elevation: 12
        width: "240dp"

    MDNavigationDrawerMenu:

        MDBoxLayout:
            orientation: "vertical"
            adaptive_height: True
            spacing: "12dp"
            padding: 0, 0, 0, "12dp"

            MDIconButton:
                icon: "menu"

            ExtendedButton:
                text: "Compose"
                icon: "pencil"

```

(continues on next page)

(continued from previous page)

```

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

        DrawerClickableItem:
            text: "JavaScript"
            icon: "language-javascript"

        DrawerClickableItem:
            text: "CPP"
            icon: "language-cpp"

        DrawerClickableItem:
            text: "Swift"
            icon: "language-swift"
'''

class ExtendedButton(
    RoundedRectangularElevationBehavior, MDFillRoundFlatIconButton
):
    '''
    Implements a button of type
    `Extended FAB <https://m3.material.io/components/extended-fab/overview>`_.

    .. rubric::
        Extended FABs help people take primary actions.
        They're wider than FABs to accommodate a text label and larger target
        area.

    This type of buttons is not yet implemented in the standard widget set
    of the KivyMD library, so we will implement it ourselves in this class.
    '''

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.padding = "16dp"
        Clock.schedule_once(self.set_spacing)

    def set_spacing(self, interval):
        self.ids.box.spacing = "12dp"

    def set_radius(self, *args):
        if self.rounded_button:
            self._radius = self.radius = self.height / 4

class Example(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        self.theme_cls.primary_palette = "Orange"

```

(continues on next page)

(continued from previous page)

```

return Builder.load_string(KV)

def switch_screen(
    self, instance_navigation_rail, instance_navigation_rail_item
):
    '''
    Called when tapping on rail menu items. Switches application screens.
    '''

    self.root.ids.screen_manager.current = (
        instance_navigation_rail_item.icon.split("-")[1].lower()
    )

def on_start(self):
    '''Creates application screens.'''

    navigation_rail_items = self.root.ids.navigation_rail.get_items()[:]
    navigation_rail_items.reverse()

    for widget in navigation_rail_items:
        name_screen = widget.icon.split("-")[1].lower()
        screen = MDScreen(
            name=name_screen,
            md_bg_color="#edd769",
            radius=[18, 0, 0, 0],
        )
        box = MDBoxLayout(padding="12dp")
        label = MDLabel(
            text=name_screen.capitalize(),
            font_style="H1",
            halign="right",
            adaptive_height=True,
            shorten=True,
        )
        box.add_widget(label)
        screen.add_widget(box)
        self.root.ids.screen_manager.add_widget(screen)

```

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

API - kivymd.uix.navigationrail.navigationrail**class** kivymd.uix.navigationrail.navigationrail.**MDNavigationRailFabButton**(**kwargs)

Implements an optional floating action button (FAB).

icon

Button icon name.

```
MDNavigationRail:  
    MDNavigationRailFabButton:  
        icon: "home"
```

*icon* is an `StringProperty` and defaults to `'pencil'`.**class** kivymd.uix.navigationrail.navigationrail.**MDNavigationRailMenuButton**(**kwargs)

Implements a menu button.

icon

Button icon name.

```
MDNavigationRail:
```

```
    MDNavigationRailMenuButton:
```

```
        icon: "home"
```



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

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

Implements a menu item with an icon and text.

navigation_rail

MDNavigationRail object.

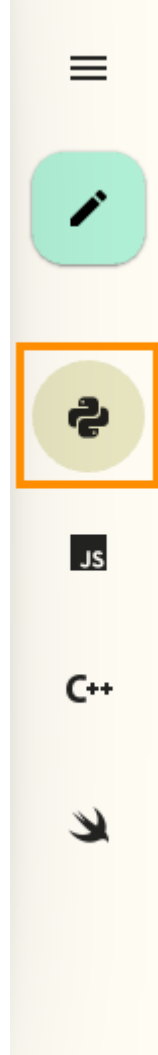
navigation_rail is an `ObjectProperty` and defaults to `None`.

icon

Icon item.

```
MDNavigationRail:

    MDNavigationRailItem:
        icon: "language-python"
```



icon is an `StringProperty` and defaults to `'checkbox-blank'`.

text

Text item.

```
MDNavigationRail:

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



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

badge_icon

Badge icon name.

```
MDNavigationRail:

    MDNavigationRailItem:
        text: "Python"
        icon: "language-python"
        badge_icon: "plus"
```



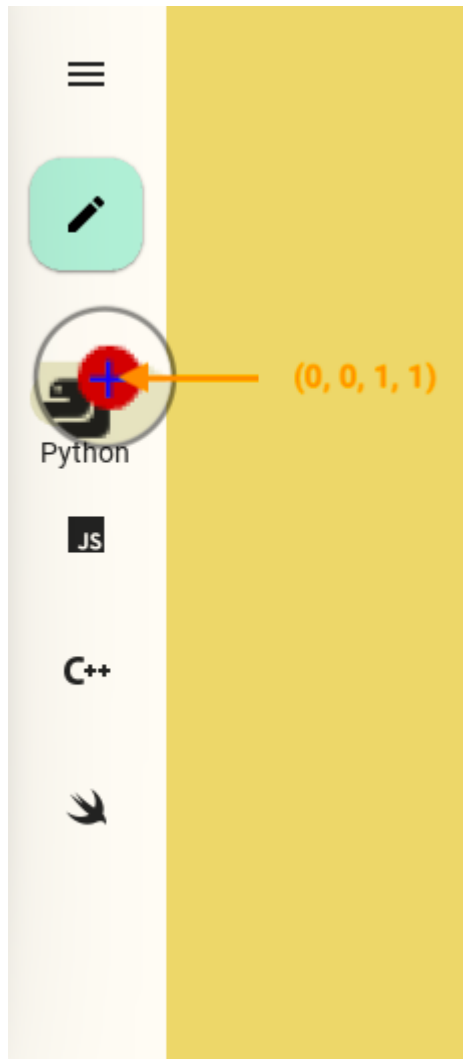
`badge_icon` is an `StringProperty` and defaults to `''`.

badge_icon_color

Badge icon color in (r, g, b, a) format.

```
MDNavigationRail:

    MDNavigationRailItem:
        text: "Python"
        icon: "language-python"
        badge_icon: "plus"
        badge_icon_color: 0, 0, 1, 1
```

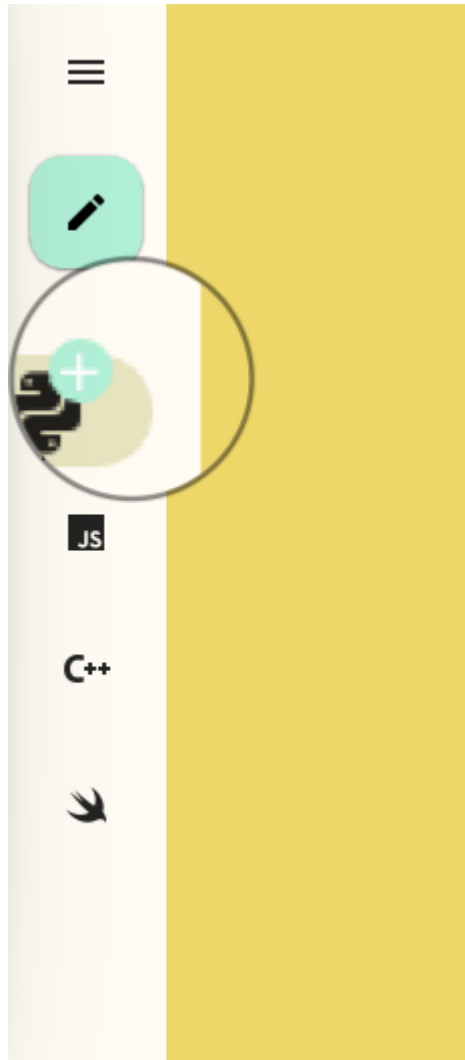


`badge_icon_color` is an `StringProperty` and defaults to `None`.

badge_bg_color

Badge icon background color in (r, g, b, a) format.

```
MDNavigationRail:
    MDNavigationRailItem:
        text: "Python"
        icon: "language-python"
        badge_icon: "plus"
        badge_bg_color: "#b0f0d6"
```

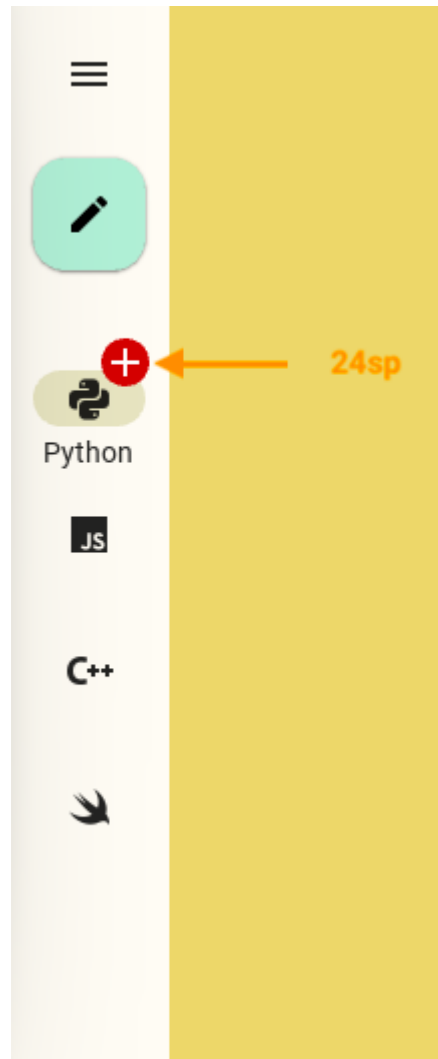


badge_bg_color is an `ColorProperty` and defaults to *None*.

badge_font_size

Badge icon font size.

```
MDNavigationRail:
    MDNavigationRailItem:
        text: "Python"
        icon: "language-python"
        badge_icon: "plus"
        badge_font_size: "24sp"
```

`badge_font_size` is an `NumericProperty` and defaults to `0`.

active

Is the element active.

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

on_active(*self*, *instance_navigation_rail_item*, *value_active*: *bool*)

Called when the value of `active` changes.

animation_size_ripple_area(*self*, *value*: *int*)

Animates the size/fade of the ripple area.

on_press(*self*)

Called when pressed on a panel element.

on_release(*self*)

Called when released on a panel element.

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

Events

`on_item_press`

Called on the `on_press` event of menu item - `MDNavigationRailItem`.

`on_item_release`

Called on the `on_release` event of menu item - `MDNavigationRailItem`.

`radius`

Rail radius.

`radius` is an `VariableListProperty` and defaults to `[0, 0, 0, 0]`.

`padding`

Padding between layout box and children: `[padding_left, padding_top, padding_right, padding_bottom]`.

`padding` is a `VariableListProperty` and defaults to `[0, '36dp', 0, '36dp']`.

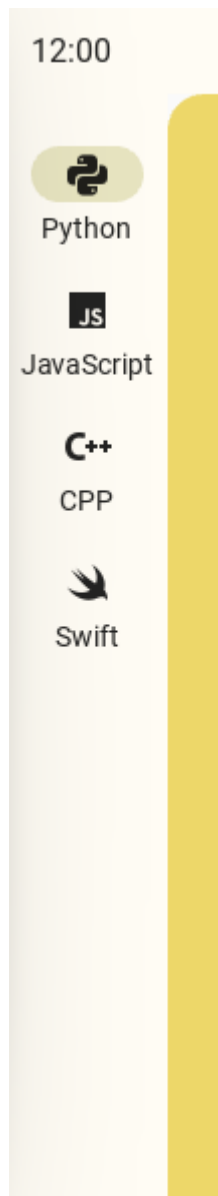
`anchor`

The position of the panel with menu items. Available options are: `'top'`, `'bottom'`, `'center'`.

Top

```
MDNavigationRail:
    anchor: "top"

    MDNavigationRailItem:
        ...
```



Center

```
MDNavigationRail:  
    anchor: "center"  
  
    MDNavigationRailItem:  
        ...
```



Bottom

```
MDNavigationRail:  
    anchor: "bottom"  
  
    MDNavigationRailItem:  
        ...
```



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

type

Type of switching menu items. Available options are: `'labeled'`, `'selected'`, `'unselected'`.

Labeled

```
MDNavigationRail:
    type: "labeled"

    MDNavigationRailItem:
        ...
```

**Selected**

```
MDNavigationRail:
    type: "selected"

    MDNavigationRailItem:
        ...
```

Unselected

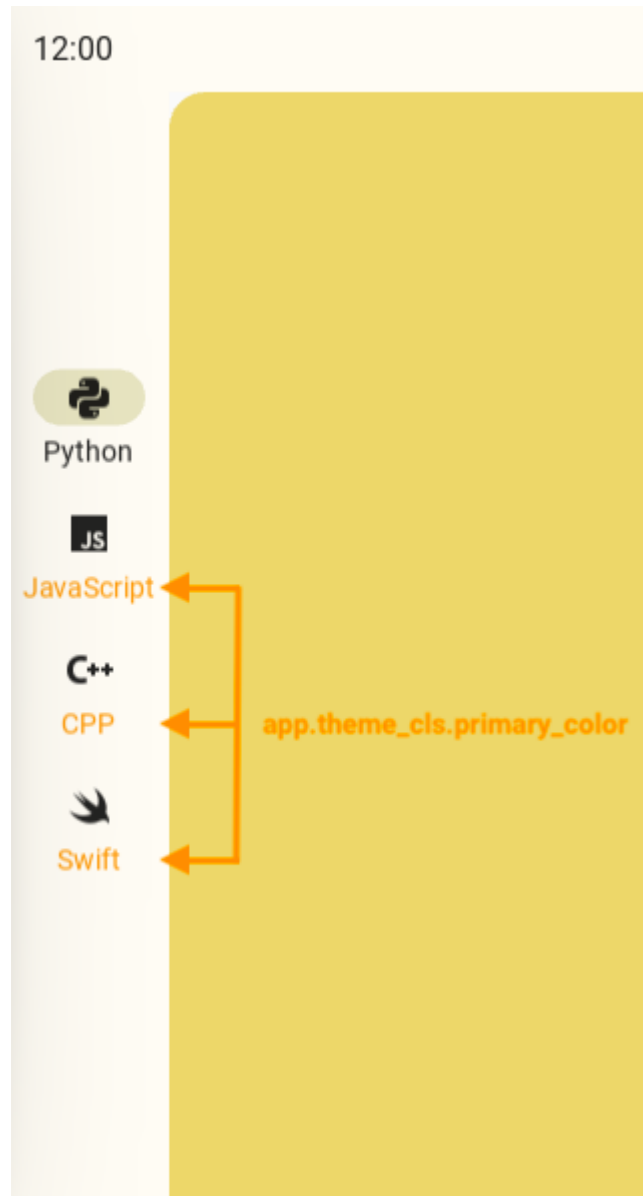
```
MDNavigationRail:  
    type: "unselected"  
  
    MDNavigationRailItem:  
        ...
```

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

text_color_item_normal

The text color of the normal menu item (*MDNavigationRailItem*).

```
MDNavigationRail:  
    text_color_item_normal: app.theme_cls.primary_color  
  
    MDNavigationRailItem:  
        ...
```



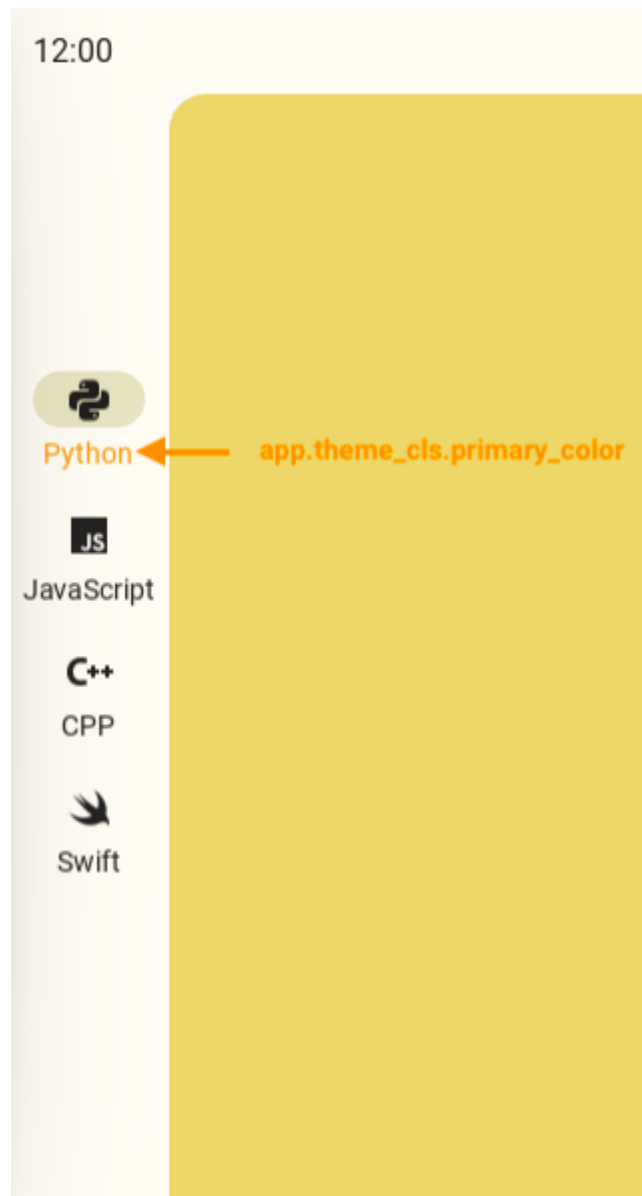
`text_color_item_normal` is a `ColorProperty` and defaults to `None`.

text_color_item_active

The text color of the active menu item (`MDNavigationRailItem`).

```
MDNavigationRail:
    text_color_item_active: app.theme_cls.primary_color

MDNavigationRailItem:
    ...
```

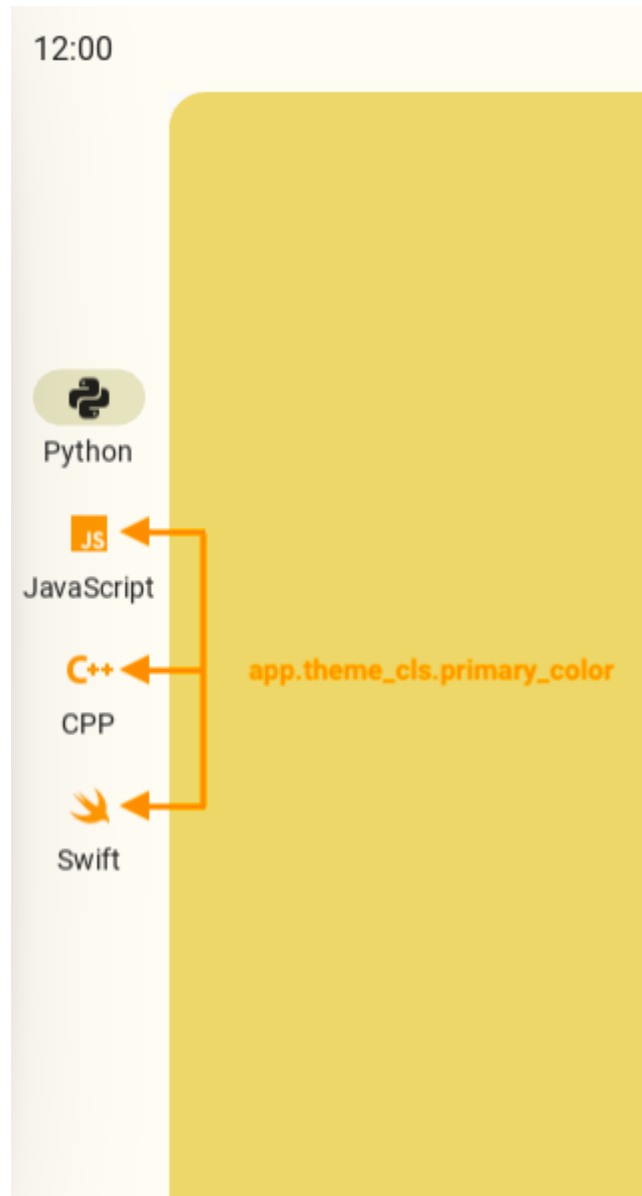
`text_color_item_active` is a `ColorProperty` and defaults to `None`.

`icon_color_item_normal`

The icon color of the normal menu item (`MDNavigationRailItem`).

```
MDNavigationRail:
    icon_color_item_normal: app.theme_cls.primary_color

MDNavigationRailItem:
    ...
```



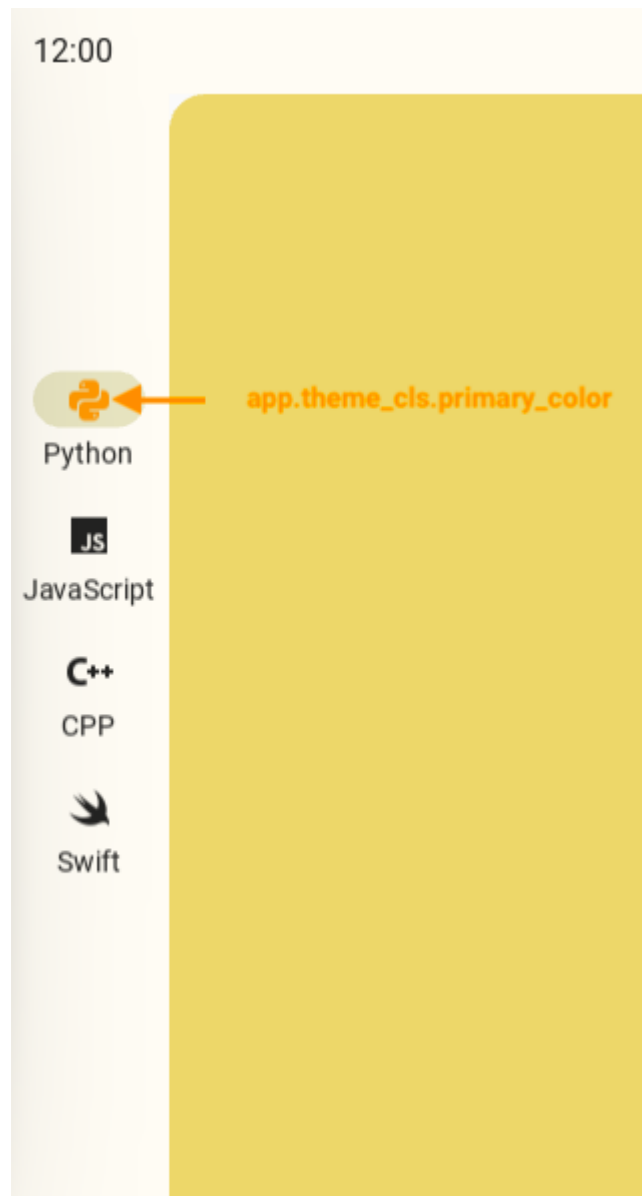
`icon_color_item_normal` is a `ColorProperty` and defaults to `None`.

icon_color_item_active

The icon color of the active menu item (`MDNavigationRailItem`).

```
MDNavigationRail:
    icon_color_item_active: app.theme_cls.primary_color

MDNavigationRailItem:
    ...
```



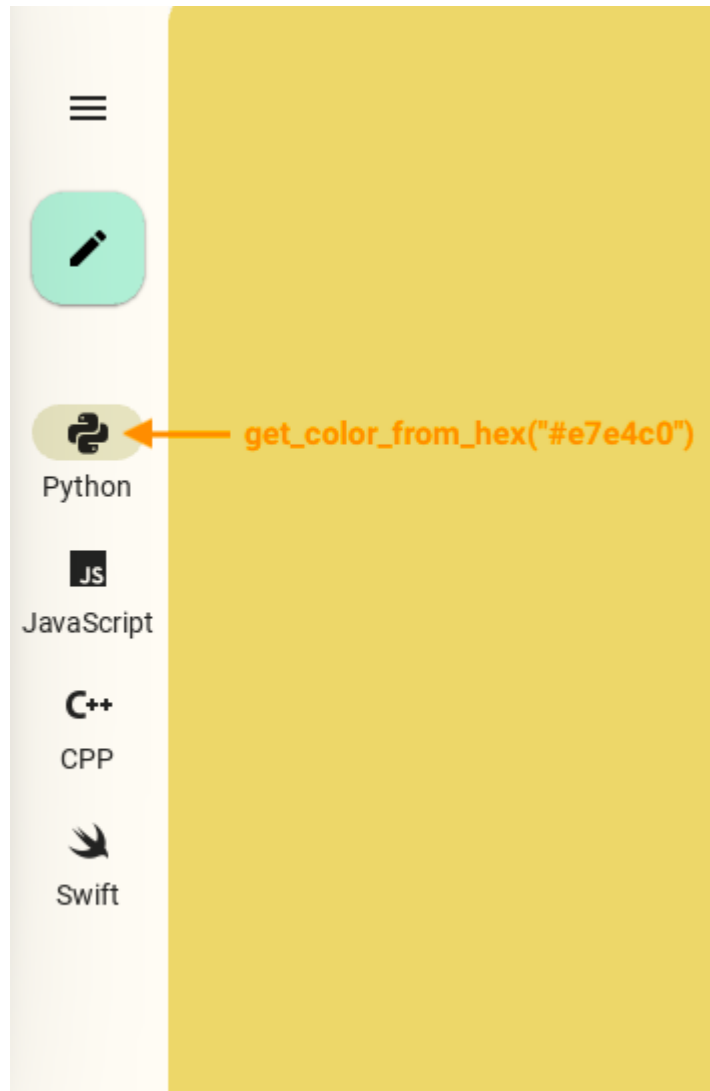
icon_color_item_active is an `ColorProperty` and defaults to *None*.

selected_color_background

Background color which will highlight the icon of the active menu item - *MDNavigationRailItem* - in (r, g, b, a) format.

```
MDNavigationRail:
    selected_color_background: "#e7e4c0"

MDNavigationRailItem:
    ...
```



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

ripple_color_item

Ripple effect color of menu items (`MDNavigationRailItem`) in (r, g, b, a) format.

```
MDNavigationRail:
    ripple_color_item: "#e7e4c0"

MDNavigationRailItem:
    ...
```



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

ripple_transition

Type of animation of the ripple effect when a menu item is selected.

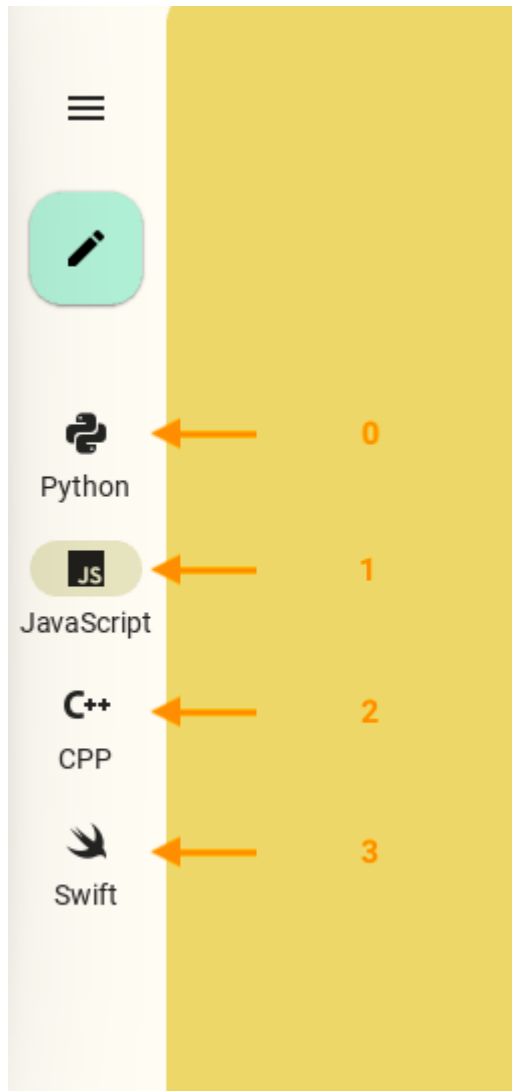
`ripple_transition` is a `StringProperty` and defaults to `'ripple_transition'`.

current_selected_item

Index of the menu list item (`MENavigationRailItem`) that will be active by default

```
MDNavigationRail:
    current_selected_item: 1

    MDNavigationRailItem:
        ...
```



`current_selected_item` is a `NumericProperty` and defaults to `0`.

font_name

Font path for menu item (`MDNavigationRailItem`) text.

```
MDNavigationRail:
    MDNavigationRailItem:
        text: "Python"
        icon: "language-python"
        font_name: "nasalization-rg.ttf"
```



`font_name` is an `StringProperty` and defaults to `'Roboto'`.

on_item_press(*self*, *args)

Called on the `on_press` event of menu item - `MDNavigationRailItem`.

on_item_release(*self*, *args)

Called on the `on_release` event of menu item - `MDNavigationRailItem`.

deselect_item(*self*, *selected_navigation_rail_item*: `MDNavigationRailItem`)

Sets the *active* value to *False* for all menu items (`MDNavigationRailItem`) except the selected item.
Called when a menu item is touched.

get_items(*self*)

Returns a list of `MDNavigationRailItem` objects

set_pos_panel_items(*self*, *instance_fab_button*: `Union[None, MDNavigationRailFabButton]`,
instance_menu_button: `Union[None, MDNavigationRailFabButton]`)

Set `PanelItems` panel position with menu items.

set_current_selected_item(*self*, *interval*: `Union[int, float]`)

Sets the active menu list item (`MDNavigationRailItem`).

set_pos_menu_fab_buttons(*self*, *interval*: *Union[int, float]*)

Sets the position of the *MDNavigationRailFabButton* and *MDNavigationRailMenuButton* buttons on the panel.

add_widget(*self*, *widget*, **args*, ***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.

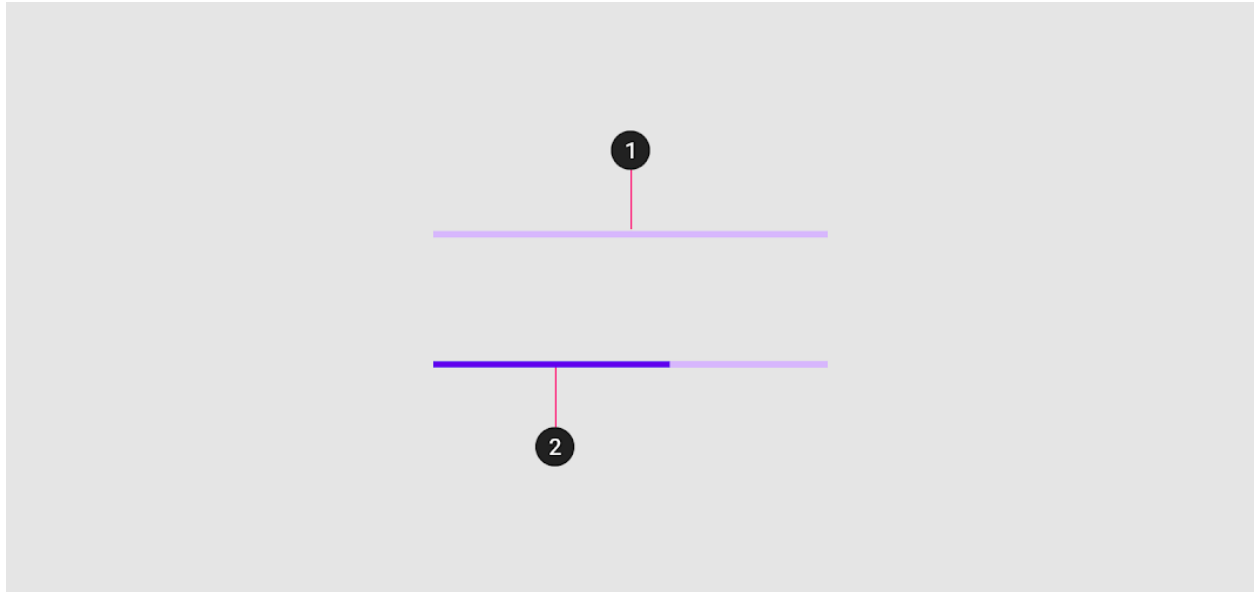
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.28 ProgressBar

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

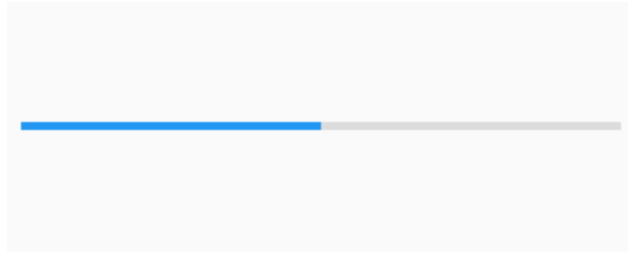
from kivymd.app import MDApp

KV = '''
MDBoxLayout:
    padding: "10dp"

    MDProgressBar:
        value: 50
'''

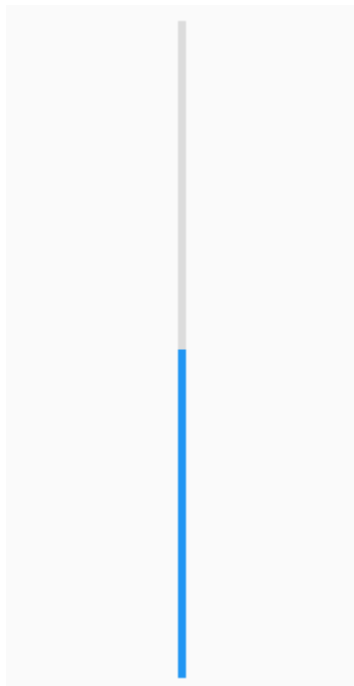
class Test(MDApp):
    def build(self):
        return Builder.load_string(KV)

Test().run()
```



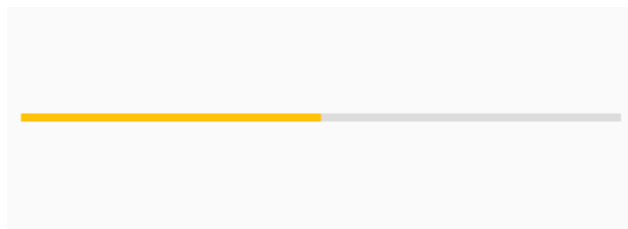
Vertical orientation

```
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 = '''
MDScreen:

    MDProgressBar:
        id: progress
        pos_hint: {"center_y": .6}
        type: "indeterminate"

    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.progressbar**class** kivymd.uix.progressbar.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*.**back_color**

Progress bar back color in rgba format.

New in version 1.0.0.

back_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.

running_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*.**check_size**(self, interval: *Union[int, float]*)**start**(self)

Start animation.

stop(self)

Stop animation.

```
running_away(self, *args)
```

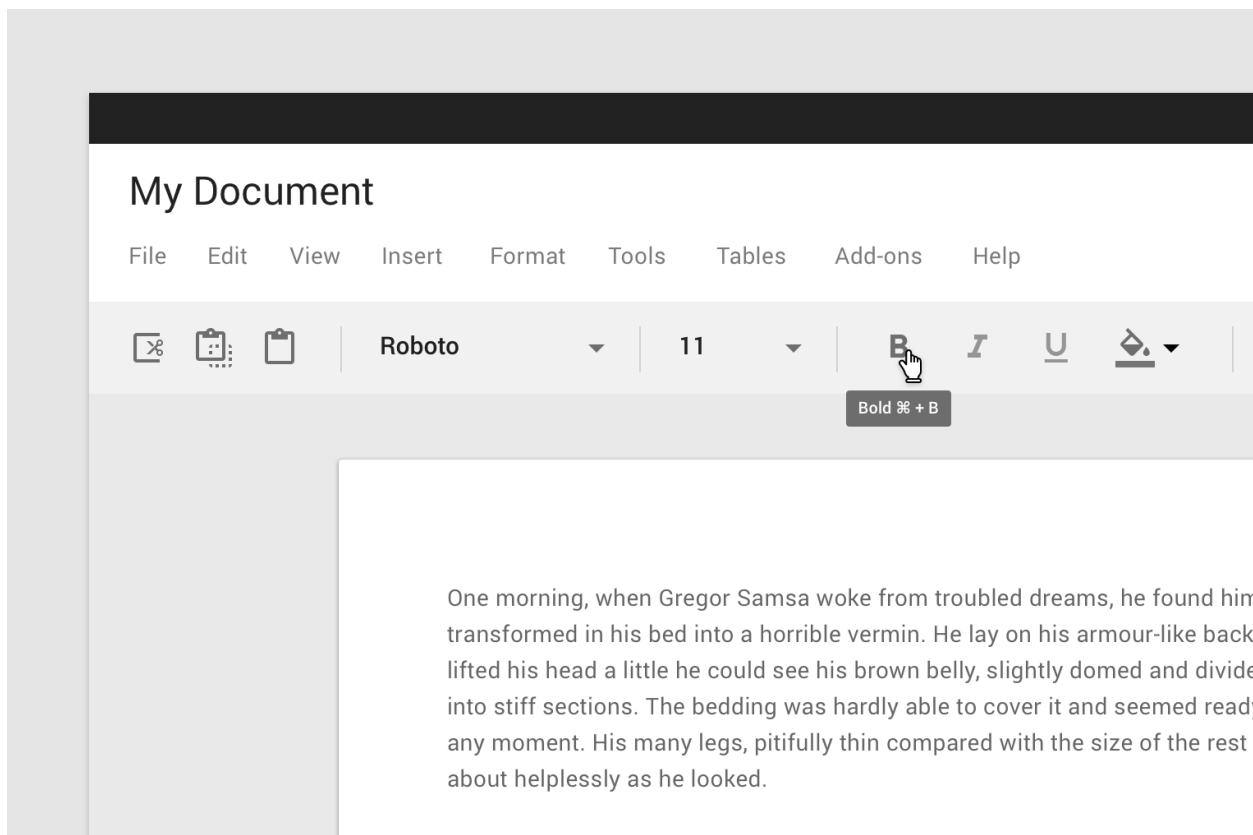
```
catching_up(self, *args)
```

2.3.29 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>

MDScreen:

    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.tooltip`

class `kivymd.uix.tooltip.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 `NumericProperty` and defaults to `0`.

shift_right

Shifting the tooltip text to the right.

New in version 1.0.0.

`shift_right` is an `NumericProperty` and defaults to `0`.

shift_left

Shifting the tooltip text to the left.

New in version 1.0.0.

`shift_left` is an `NumericProperty` and defaults to `0`.

delete_clock(*self*, *widget*, *touch*, **args*)**adjust_tooltip_position(*self*, *x*: *float*, *y*: *float*)**

Returns the coordinates of the tooltip that fit into the borders of the screen.

display_tooltip(*self*, *interval*: *Union[int, float]*)**animation_tooltip_show(*self*, *interval*: *Union[int, float]*)**

Animation of opening tooltip on the screen.

animation_tooltip_dismiss(*self*, *interval*: *Union[int, float]*)

New in version 1.0.0.

Animation of closing tooltip on the screen.

remove_tooltip(*self*, **args*)

Removes the tooltip widget from the screen.

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*)

Default dismiss event handler.

on_dismiss(*self*)

New in version 1.0.0.

Default dismiss event handler.

class kivymd.uix.tooltip.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

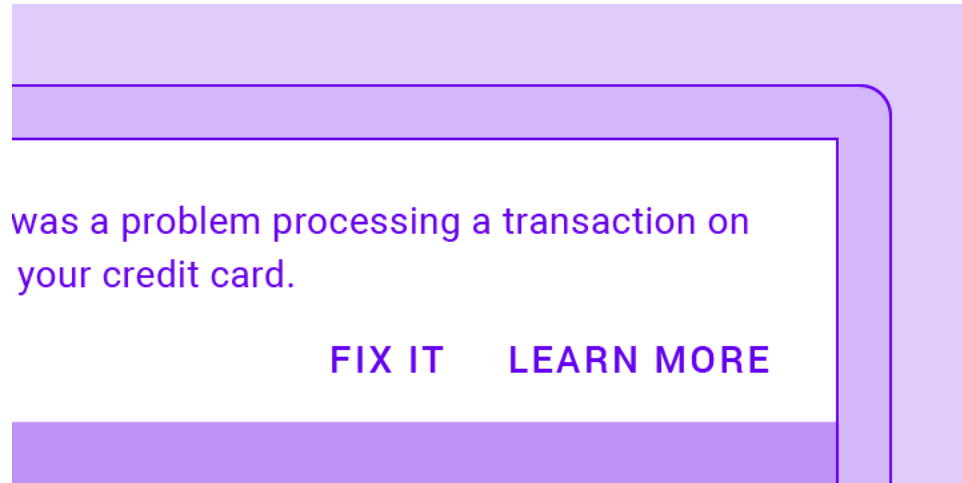
See [tooltip_radius](#).

2.3.30 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

    MDTopAppBar:
        id: toolbar
        title: "Example Banners"
        elevation: 10
        pos_hint: {'top': 1}

    MDBoxLayout:
        id: screen
        orientation: "vertical"
        size_hint_y: None
        height: Window.height - toolbar.height

    OneLineListItem:
        text: "Banner without actions"
        on_release: banner.show()
```

(continues on next page)

(continued from previous page)

```

'''
Widget:

class Test(MDApp):
    def build(self):
        return Factory.ExampleBanner()

Test().run()

```

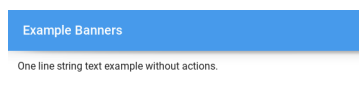
Banner type.

By default, the banner is of the type 'one-line':

```

MDBanner:
    text: ["One line string text example without actions."]

```

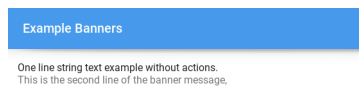


To use a two-line banner, specify the 'two-line' *MDBanner.type* for the banner and pass the list of two lines to the *MDBanner.text* parameter:

```

MDBanner:
    type: "two-line"
    text: ["One line string text example without actions.", "This is the second line of_
↪the banner message."]

```

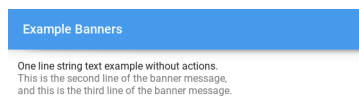


Similarly, create a three-line banner:

```

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."]

```



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]

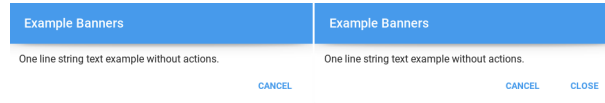
```

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.banner

class kivymd.uix.banner.banner.MDBanner(**kwargs)

Common base class for rectangular and circular elevation behavior.

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’*.

opening_timeout

Time interval after which the banner will be shown.

New in version 1.0.0.

opening_timeout is an `BoundedNumericProperty` and defaults to *0.7*.

opening_time

The time taken for the banner to slide to the state *‘open’*.

New in version 1.0.0.

opening_time is a `NumericProperty` and defaults to *0.15*.

closing_time

The time taken for the banner to slide to the state *‘close’*.

New in version 1.0.0.

closing_time is a `NumericProperty` and defaults to *0.15*.

add_actions_buttons(*self*, *instance_box*: `MDBoxLayout`, *data*: *list*)

Adds buttons to the banner.

Parameters

data – [*‘NAME BUTTON’*, <function>];

show(*self*)

Displays a banner on the screen.

hide(*self*)

Hides the banner from the screen.

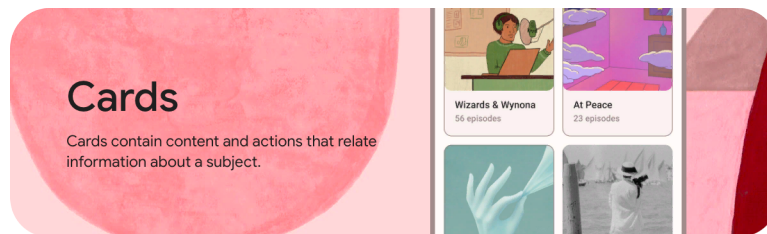
set_type_banner(*self*)**animation_display_banner**(*self*, *interval*: `Union[int, float]`)

2.3.31 Card

See also:

Material Design spec, Cards and Material Design 3 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

Warning: Starting from the KivyMD 1.0.0 library version, it is necessary to manually inherit the card class from one of the Elevation classes from `kivymd/uix/behaviors/elevation.py` module to draw the card shadow.

```
from kivymd.uix.behaviors import RoundedRectangularElevationBehavior
from kivymd.uix.card import MDCard

class MD3Card(MDCard, RoundedRectangularElevationBehavior):
    '''Implements a material design v3 card.'''
```

This may sound awkward to you, but it actually allows for better control over the providers that implement the rendering of the shadows.

Note: You can read more information about the classes that implement the rendering of shadows on this [documentation page](#).

An example of the implementation of a card in the style of material design version 3

```

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

from kivymd.app import MDApp
from kivymd.uix.behaviors import RoundedRectangularElevationBehavior
from kivymd.uix.card import MDCard

KV = '''
<MD3Card>
    padding: 16
    size_hint: None, None
    size: "200dp", "100dp"

    MDRelativeLayout:
        size_hint: None, None
        size: root.size

        MDIconButton:
            icon: "dots-vertical"
            pos:
                root.width - (self.width + root.padding[0] + dp(4)),
↪root.height - (self.height + root.padding[0] + dp(4))

        MDLabel:
            id: label
            text: root.text
            adaptive_size: True
            color: .2, .2, .2, .8

MDScreen:

    MDBoxLayout:
        id: box
        adaptive_size: True
        spacing: "56dp"
        pos_hint: {"center_x": .5, "center_y": .5}
'''

class MD3Card(MDCard, RoundedRectangularElevationBehavior):
    '''Implements a material design v3 card.'''

    text = StringProperty()

class TestCard(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(KV)

```

(continues on next page)

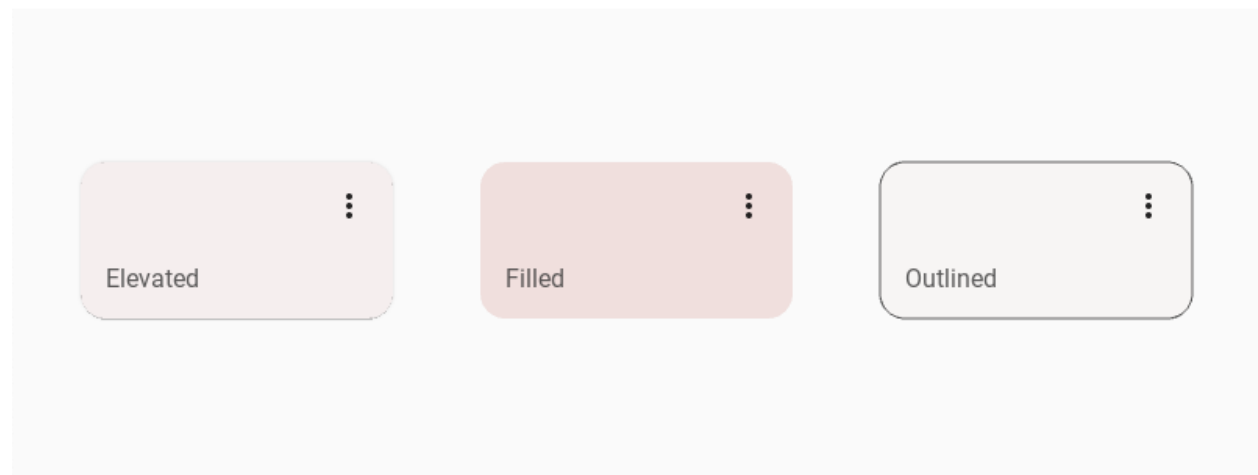
(continued from previous page)

```

def on_start(self):
    styles = {
        "elevated": "#f6eeee", "filled": "#f4dedc", "outlined": "#f8f5f4"
    }
    for style in styles.keys():
        self.root.ids.box.add_widget(
            MD3Card(
                line_color=(0.2, 0.2, 0.2, 0.8),
                style=style,
                text=style.capitalize(),
                md_bg_color=styles[style],
            )
        )

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:

```

(continues on next page)

(continued from previous page)

```

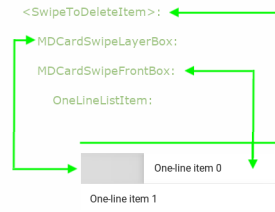
OneLineListItem:
    id: content
    text: root.text
    _no_ripple_effect: True

```

```

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:

    MDBoxLayout:
        orientation: "vertical"
        spacing: "10dp"

        MDTopAppBar:
            elevation: 10

```

(continues on next page)

(continued from previous page)

```

        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):
        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:
```

(continues on next page)

(continued from previous page)

```

MDBoxLayout:
    orientation: "vertical"
    spacing: "10dp"

    MDTopAppBar:
        elevation: 10
        title: "MDCardSwipe"

    ScrollView:

        MDList:
            id: md_list
            padding: 0
...

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"

```

(continues on next page)

(continued from previous page)

```

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:

    MDBoxLayout:
        orientation: "vertical"
        spacing: "10dp"

        MDTopAppBar:
            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 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:

    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}

        MDBoxLayout:
            id: box_top
            spacing: "20dp"
            adaptive_height: True

            FitImage:
                source: "/Users/macbookair/album.jpeg"
                size_hint: .3, None
                height: text_box.height

            MDBoxLayout:
                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
                    adaptive_height: True

                MDLabel:
                    text: "July 27, 1984"
                    adaptive_height: True
                    theme_text_color: "Primary"

    MDSeparator:
```

(continues on next page)

(continued from previous page)

```

        MDBoxLayout:
            id: box_bottom
            adaptive_height: True
            padding: "10dp", 0, 0, 0

            MDLabel:
                text: "Rate this album"
                adaptive_height: True
                pos_hint: {"center_y": .5}
                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.card`

class `kivymd.uix.card.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.card.MDCard(**kwargs)`

Common base class for rectangular and circular elevation behavior.

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.

radius

Card radius by default.

New in version 1.0.0.

`radius` is an `VariableListProperty` and defaults to `[dp(6), dp(6), dp(6), dp(6)]`.

style

Card type.

New in version 1.0.0.

Available options are: 'filled', 'elevated', 'outlined'.

`style` is an `OptionProperty` and defaults to 'elevated'.

`update_md_bg_color(self, instance_card, theme_style: str)`

`set_style(self, *args)`

`set_line_color(self)`

`set_elevation(self)`

`set_radius(self)`

`on_ripple_behavior(self, interval: Union[int, float], value_behavior: bool)`

`class kivymd.uix.card.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_swipe_to_delete_item, anchor_value: *str*)

on_open_progress(*self*, instance_swipe_to_delete_item, progress_value: *float*)

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.card.MDCardSwipeFrontBox(**kwargs)

Common base class for rectangular and circular elevation behavior.

class kivymd.uix.card.card.MDCardSwipeLayerBox(*args, **kwargs)

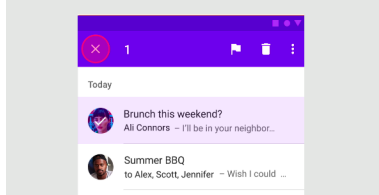
Box layout class. For more information, see in the [BoxLayout](#) class documentation.

2.3.32 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.'''

def on_unselected(self, instance_selection_list, instance_selection_item):
    '''Called when a list item is unselected.'''
```

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 MDAApp
from kivymd.uix.list import TwoLineAvatarListItem

KV = '''
<MyItem>
    text: "Two-line item with avatar"
    secondary_text: "Secondary text here"
```

(continues on next page)

(continued from previous page)

```

        _no_ripple_effect: True

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

MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
        id: toolbar
        title: "Inbox"
        left_action_items: [["menu"]]
        right_action_items: [["magnify"], ["dots-vertical"]]
        md_bg_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
            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):

```

(continues on next page)

(continued from previous page)

```

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())
    )

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.properties import ColorProperty

from kivymd.app import MDApp
from kivymd.uix.fitimage import FitImage

KV = '''
MDBoxLayout:
    orientation: "vertical"
    md_bg_color: app.theme_cls.bg_light

    MDTopAppBar:
        id: toolbar

```

(continues on next page)

(continued from previous page)

```

        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
            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 = ColorProperty("#6042e4")
    progress_round_color = "#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 = [

```

(continues on next page)

(continued from previous page)

```

        "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(
            len(instance_selection_list.get_selected_list_items())
        )

```

Example().run()

API - kivymd.uix.selection.selection

class kivymd.uix.selection.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*)

Returns True if at least one item in the list is checked.

get_selected_list_items(*self*)

Returns a list of marked objects:

[<kivymd.uix.selection.SelectionItem object>, ...]

unselected_all(*self*)

selected_all(*self*)

on_selected(*self*, *args)

Called when a list item is selected.

on_unselected(*self*, *args)

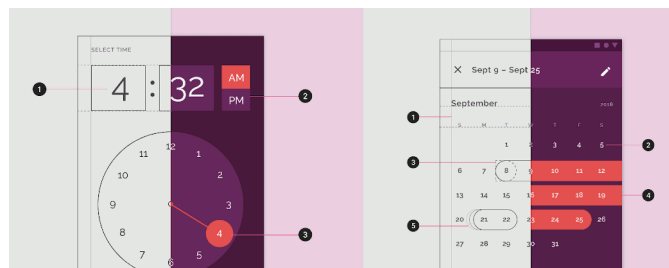
Called when a list item is unselected.

2.3.33 DatePicker

See also:

[Material Design spec, Date picker](#)

Includes date picker.



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.pickers import MDDatePicker

KV = '''
MDFloatLayout:

    MDDatePicker:
        title: "MDDatePicker"
        pos_hint: {"top": 1}
        elevation: 10
```

(continues on next page)

(continued from previous page)

```
MDRaisedButton:
    text: "Open date 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.MDDDatePicker 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)

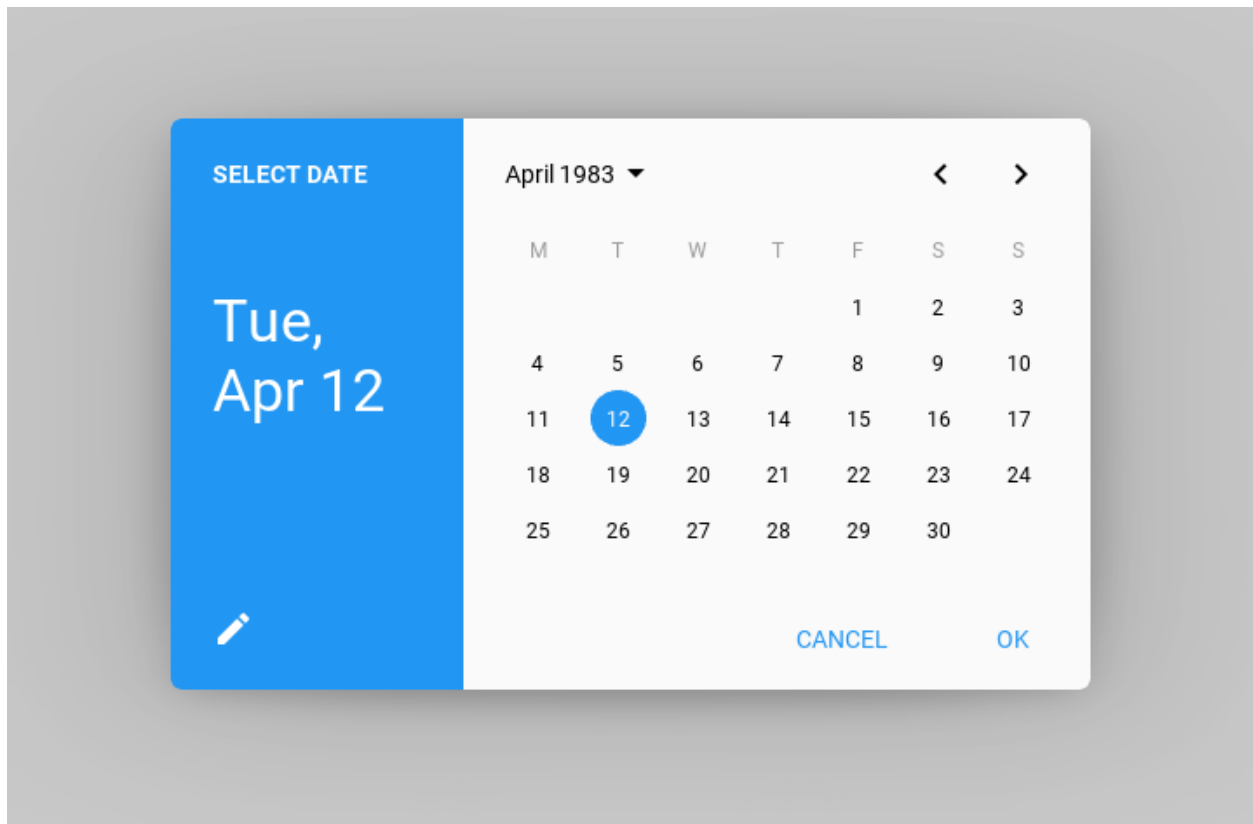
    def on_cancel(self, instance, value):
        '''Events called when the "CANCEL" dialog box button is clicked.'''

    def show_date_picker(self):
        date_dialog = MDDDatePicker()
        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 = MDDatePicker(year=1983, month=4, day=12)
    date_dialog.open()
```



Interval date

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 = MDDatePicker(
        min_date=datetime.date(2021, 2, 15),
        max_date=datetime.date(2021, 3, 27),
    )
    date_dialog.open()
```

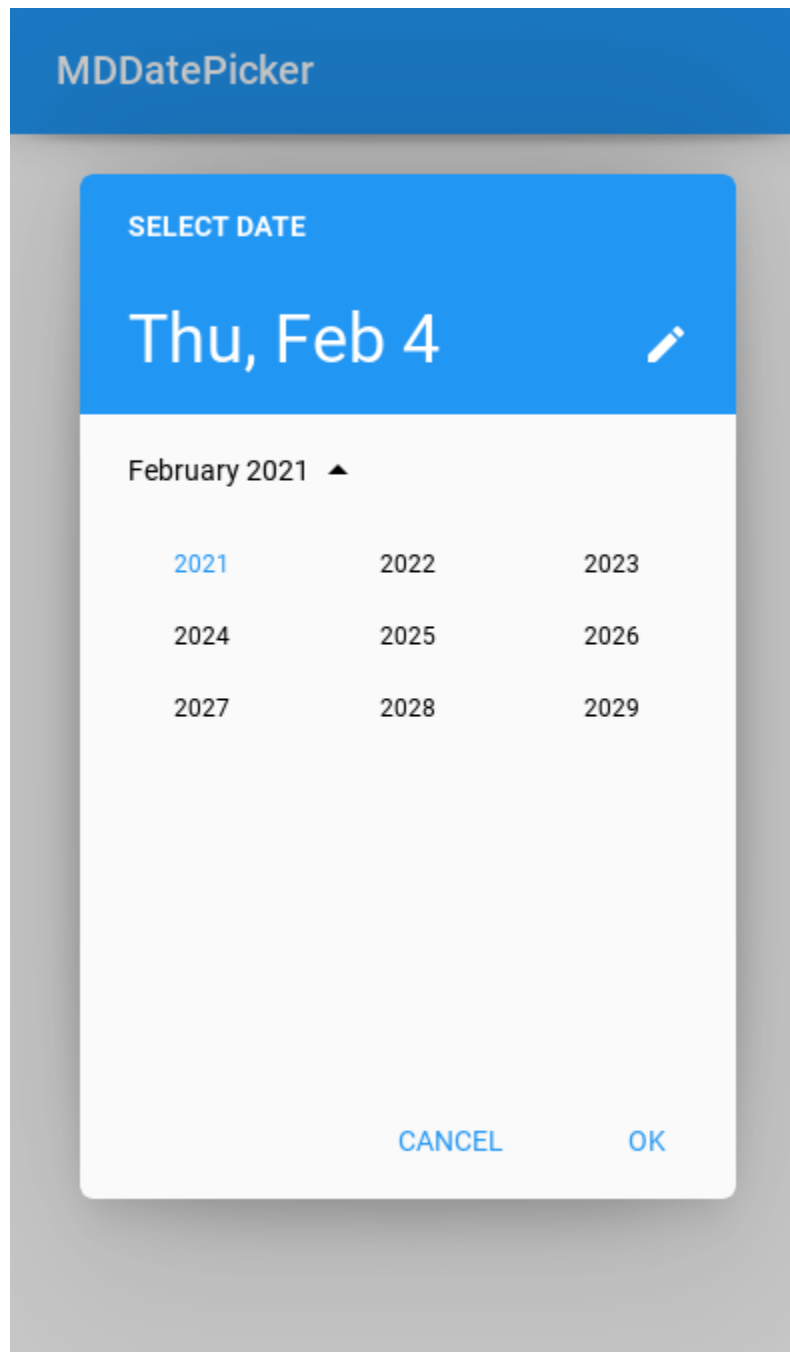
The range of available dates can be changed in the picker dialog:

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 = MDDDatePicker(mode="range")  
    date_dialog.open()
```

API - `kivymd.uix.pickers.datepicker.datepicker`

class `kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker`(***kwargs*)

Base class for `MDDDatePicker` and `MDTimePicker` classes.

Events

on_save

Events called when the “OK” dialog box button is clicked.

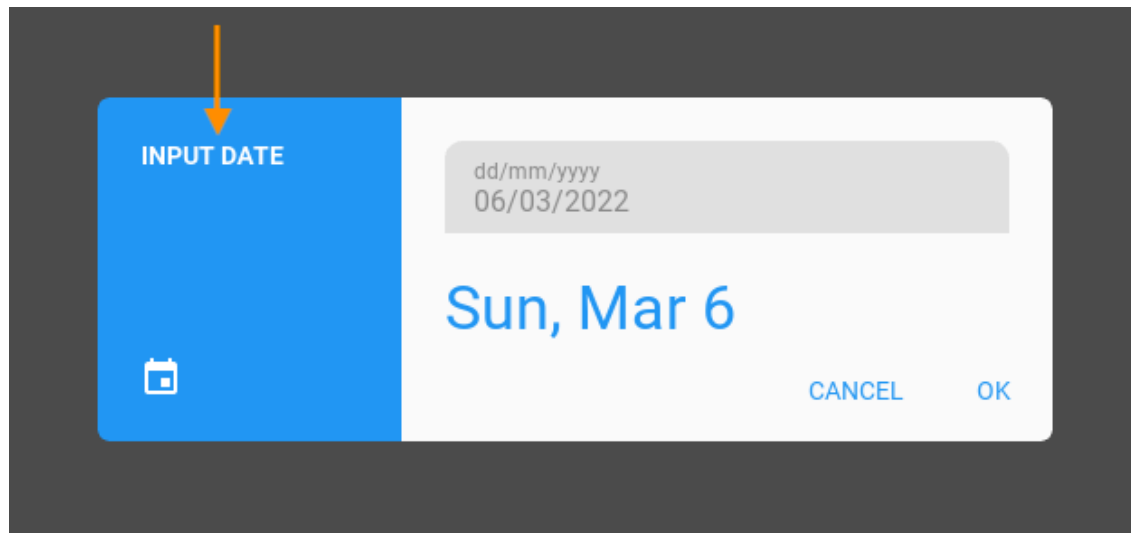
on_cancel

Events called when the “CANCEL” dialog box button is clicked.

title_input

Dialog title fot input date.

```
MDDDatePicker(title_input="INPUT DATE")
```

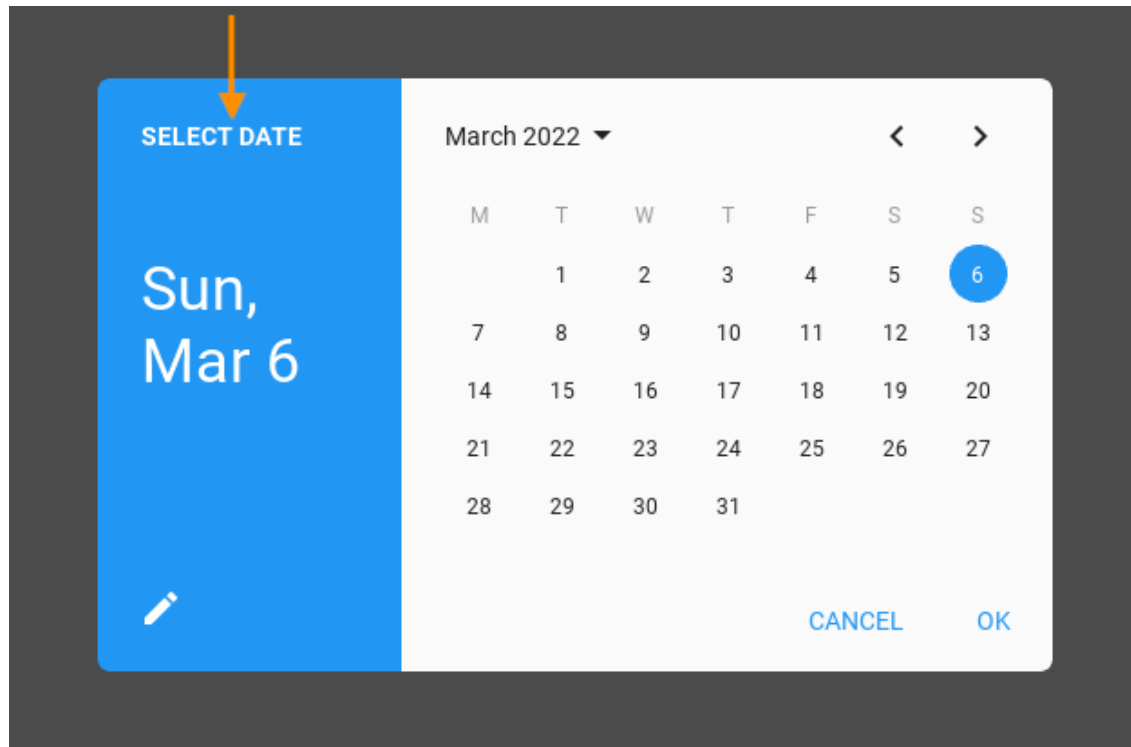


title_input is an `StringProperty` and defaults to *INPUT DATE*.

title

Dialog title fot select date.

```
MDDDatePicker(title="SELECT DATE")
```

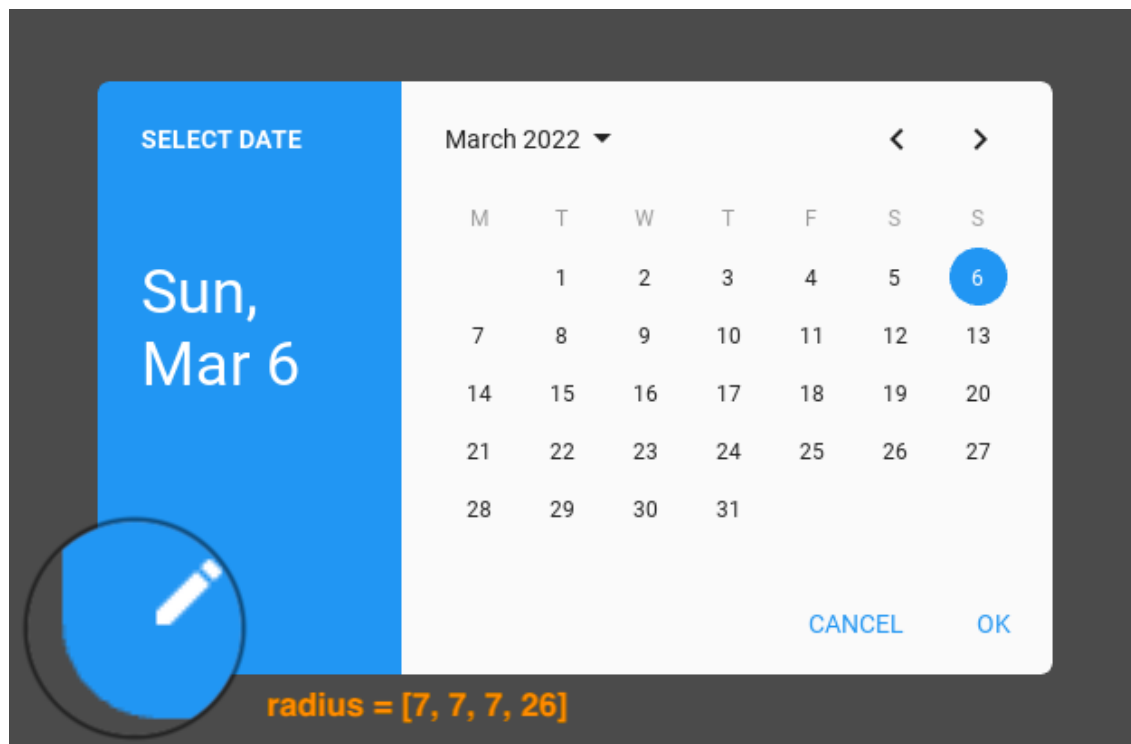


title is an `StringProperty` and defaults to `SELECT DATE`.

radius

Radius list for the four corners of the dialog.

```
MDDatePicker(radius=[7, 7, 7, 26])
```

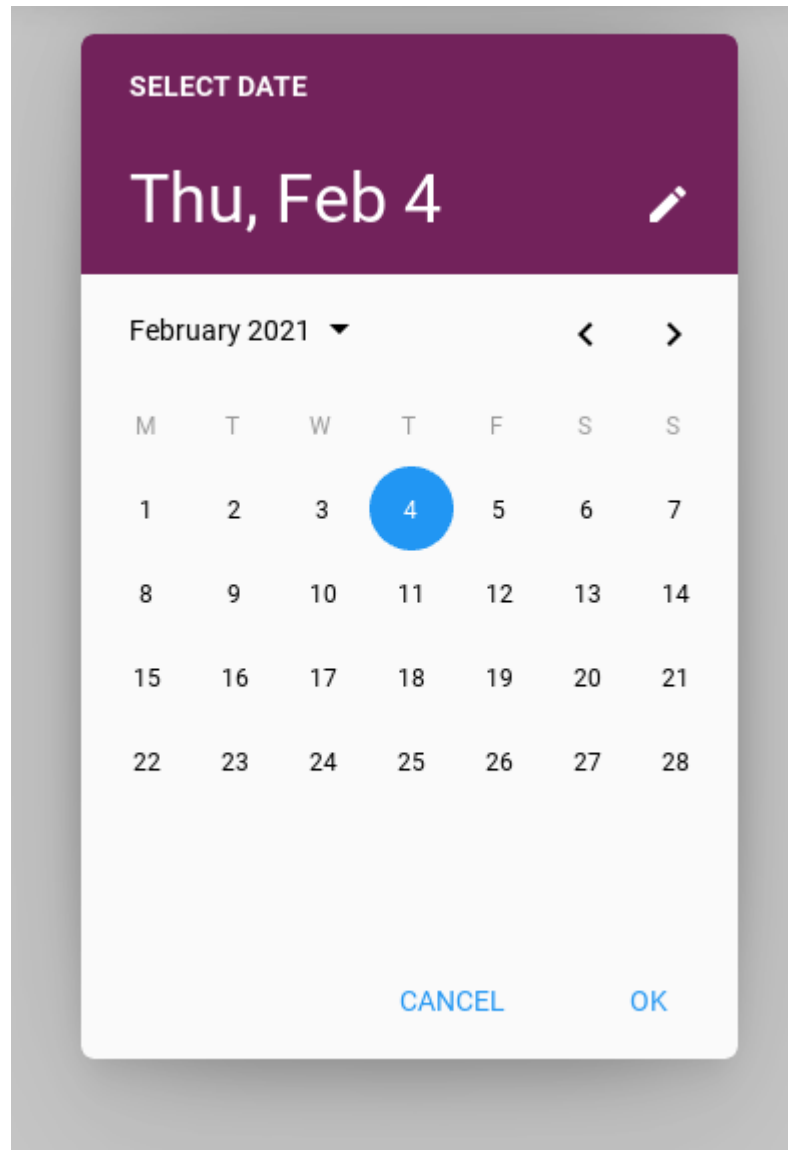


radius is an `ListProperty` and defaults to `[7, 7, 7, 7]`.

primary_color

Background color of toolbar in (r, g, b, a) format.

```
MDDatePicker(primary_color=get_color_from_hex("#72225b"))
```

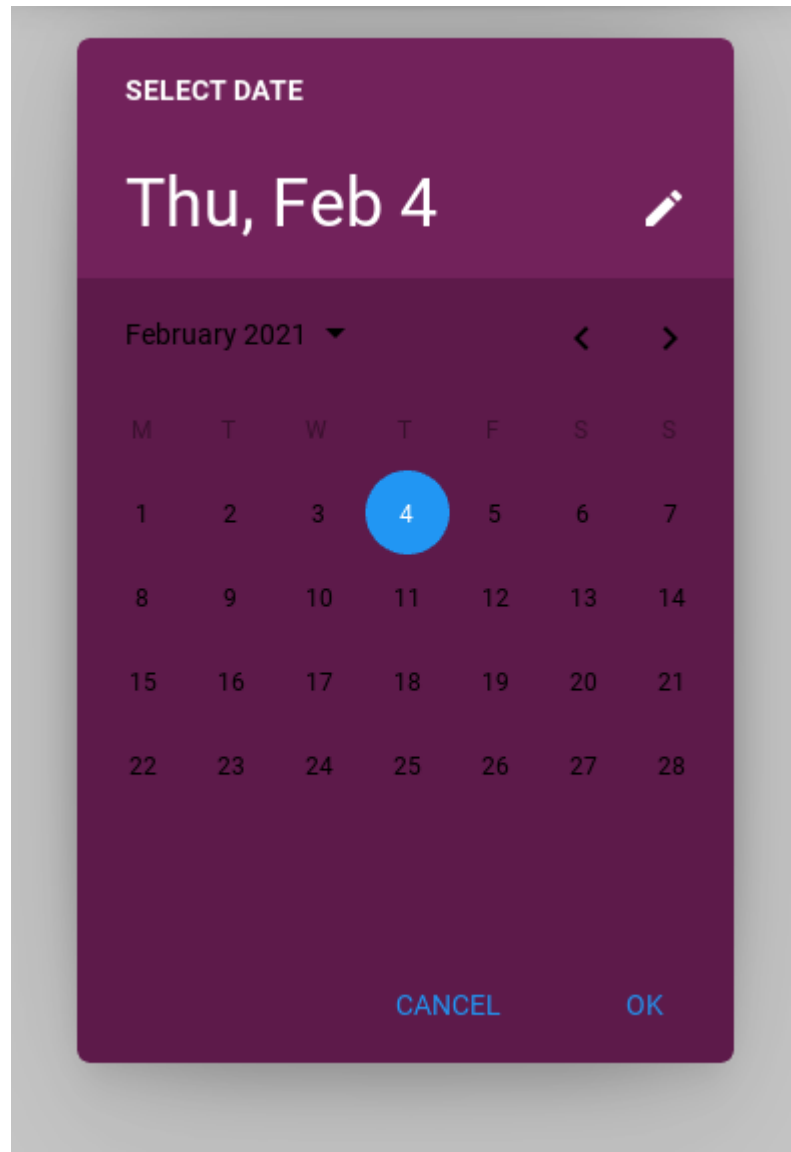


primary_color is an `ColorProperty` and defaults to `None`.

accent_color

Background color of calendar/clock face in (r, g, b, a) format.

```
MDDatePicker(
    primary_color=get_color_from_hex("#72225b"),
    accent_color=get_color_from_hex("#5d1a4a"),
)
```

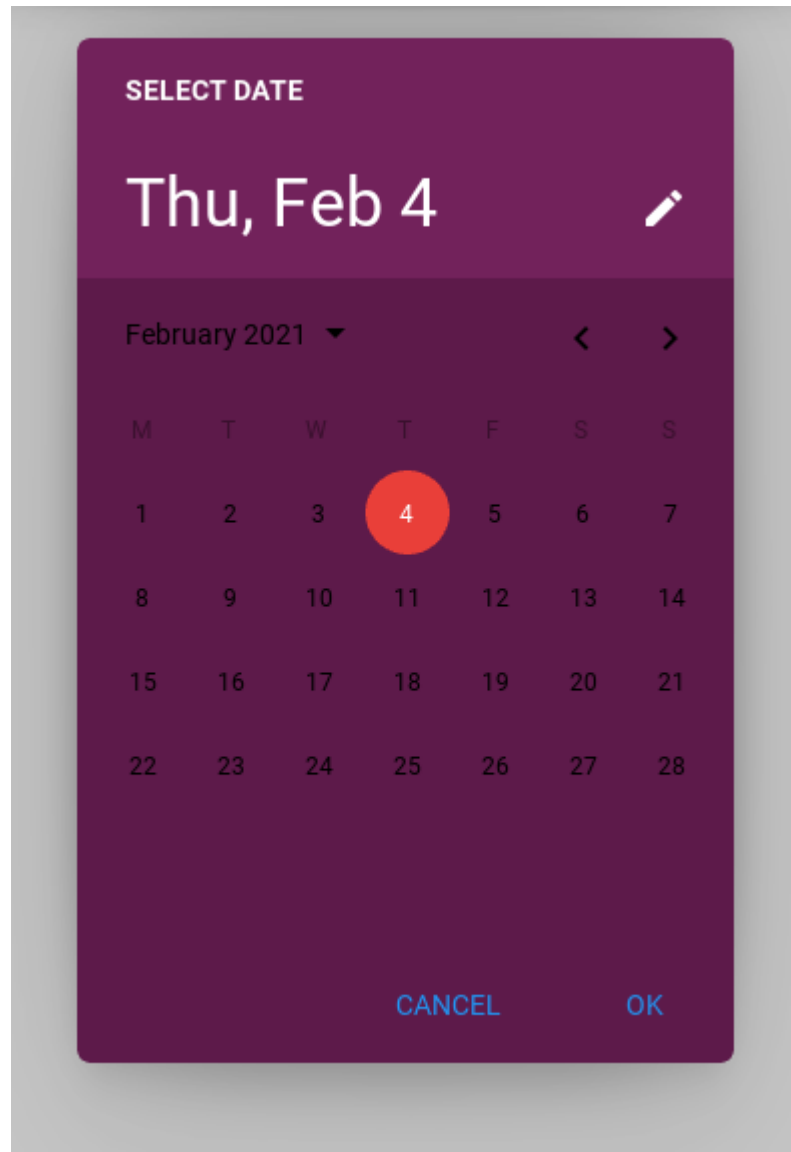



`accent_color` is an `ColorProperty` and defaults to `None`.

selector_color

Background color of the selected day of the month or hour in (r, g, b, a) format.

```
MDDatePicker(  
    primary_color=get_color_from_hex("#72225b"),  
    accent_color=get_color_from_hex("#5d1a4a"),  
    selector_color=get_color_from_hex("#e93f39"),  
)
```

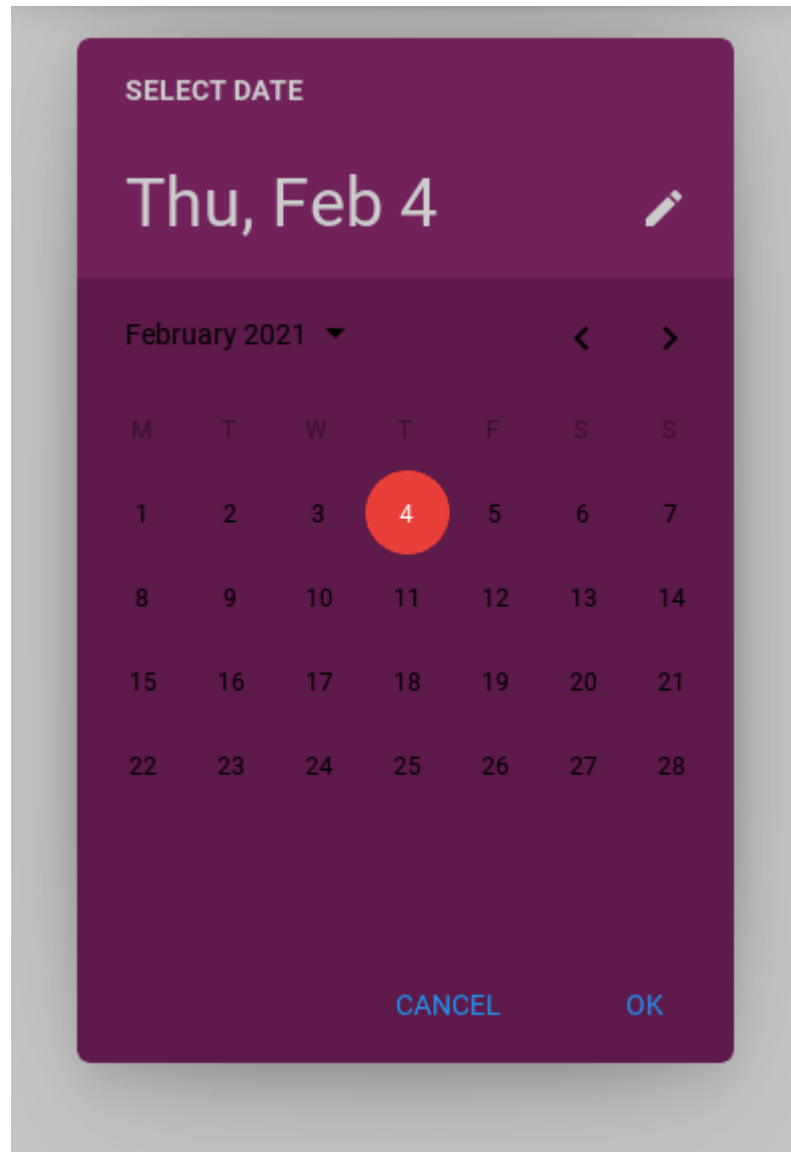


`selector_color` is an `ColorProperty` and defaults to `None`.

text_toolbar_color

Color of labels for text on a toolbar in (r, g, b, a) format.

```
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"),  
)
```

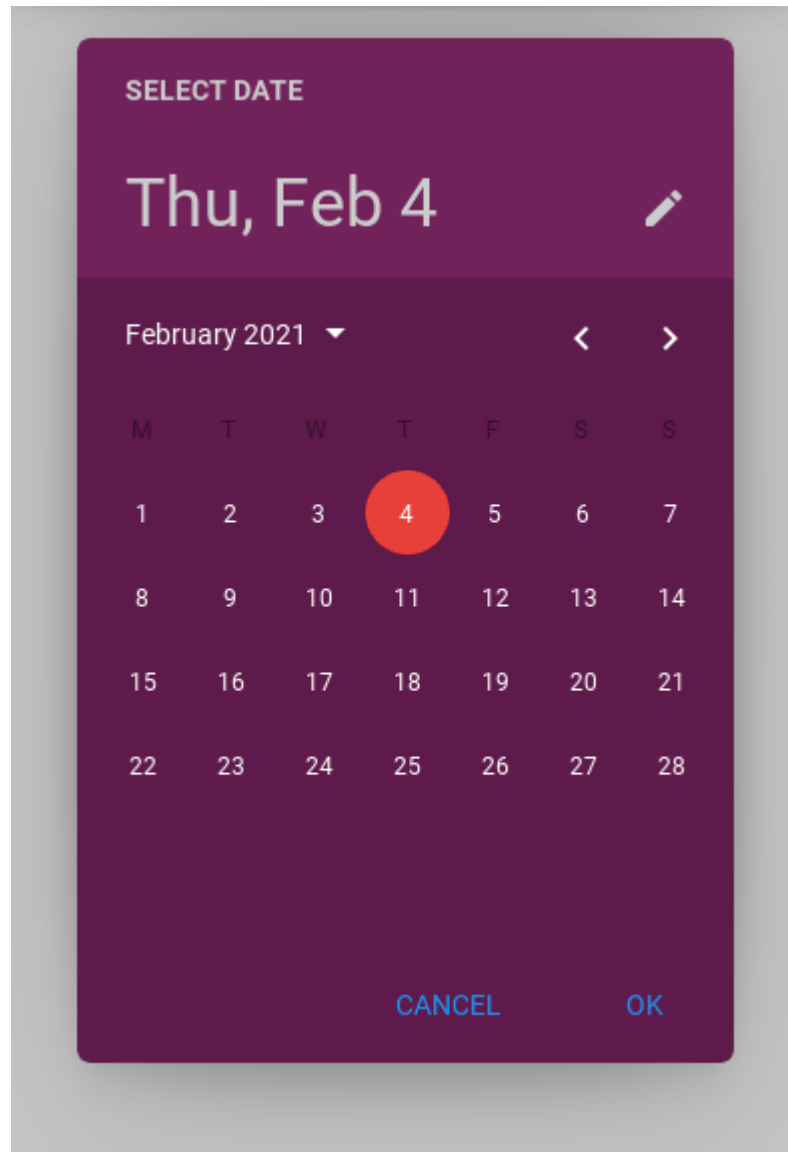


`text_toolbar_color` is an `ColorProperty` and defaults to `None`.

text_color

Color of text labels in calendar/clock face in (r, g, b, a) format.

```
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"),
)
```

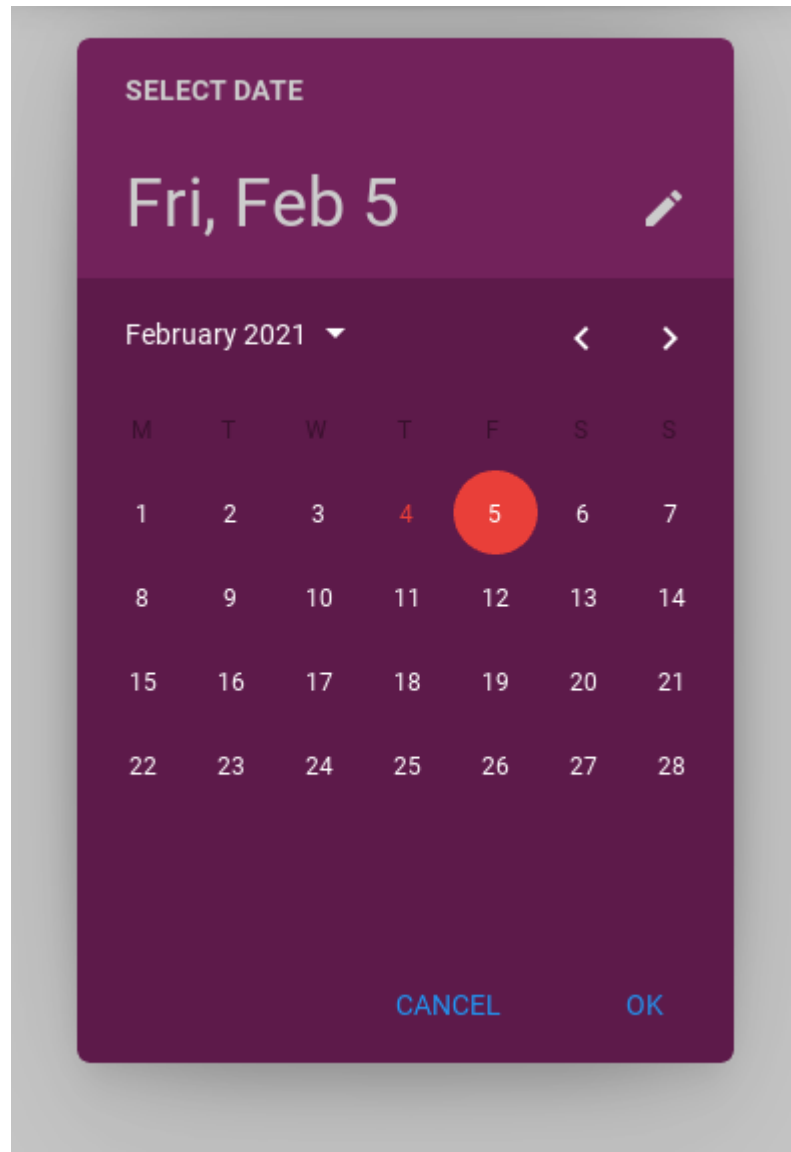


`text_color` is an `ColorProperty` and defaults to `None`.

text_current_color

Color of the text of the current day of the month/hour in (r, g, b, a) format.

```
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"),
)
```

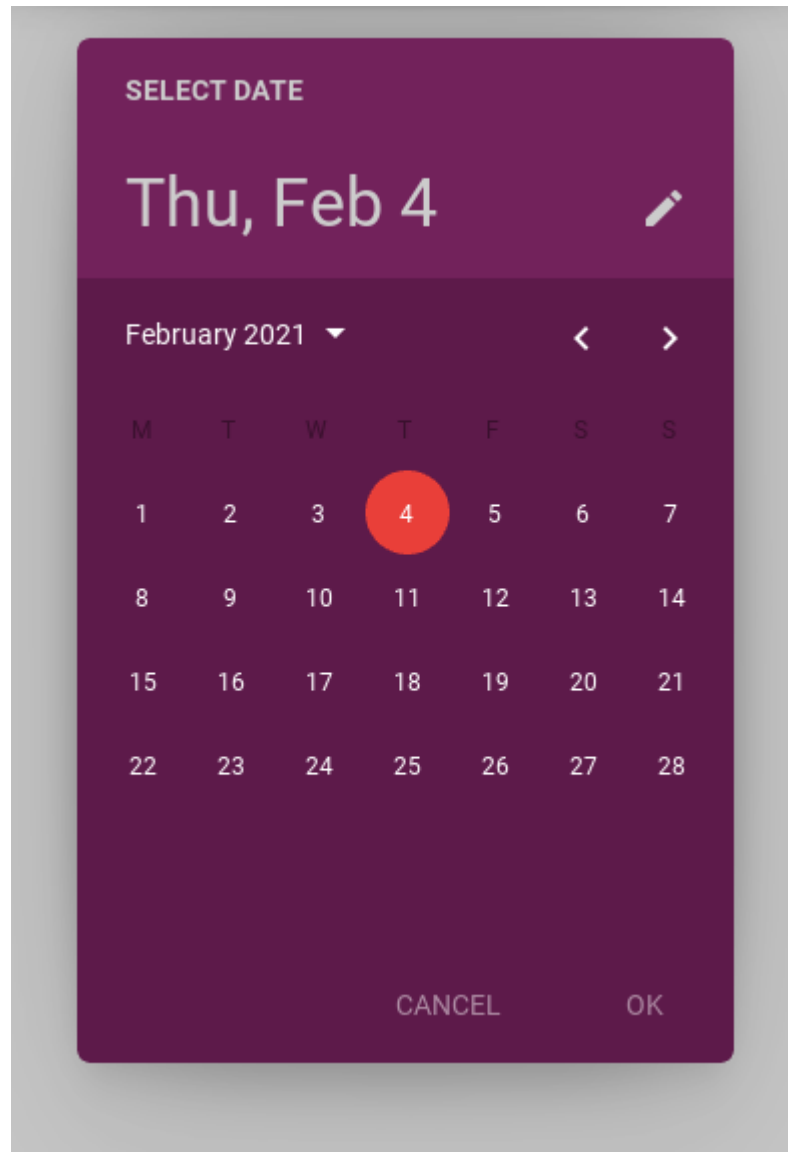


`text_current_color` is an `ColorProperty` and defaults to `None`.

text_button_color

Text button color in (r, g, b, a) format.

```
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_button_color=(1, 1, 1, .5),
)
```

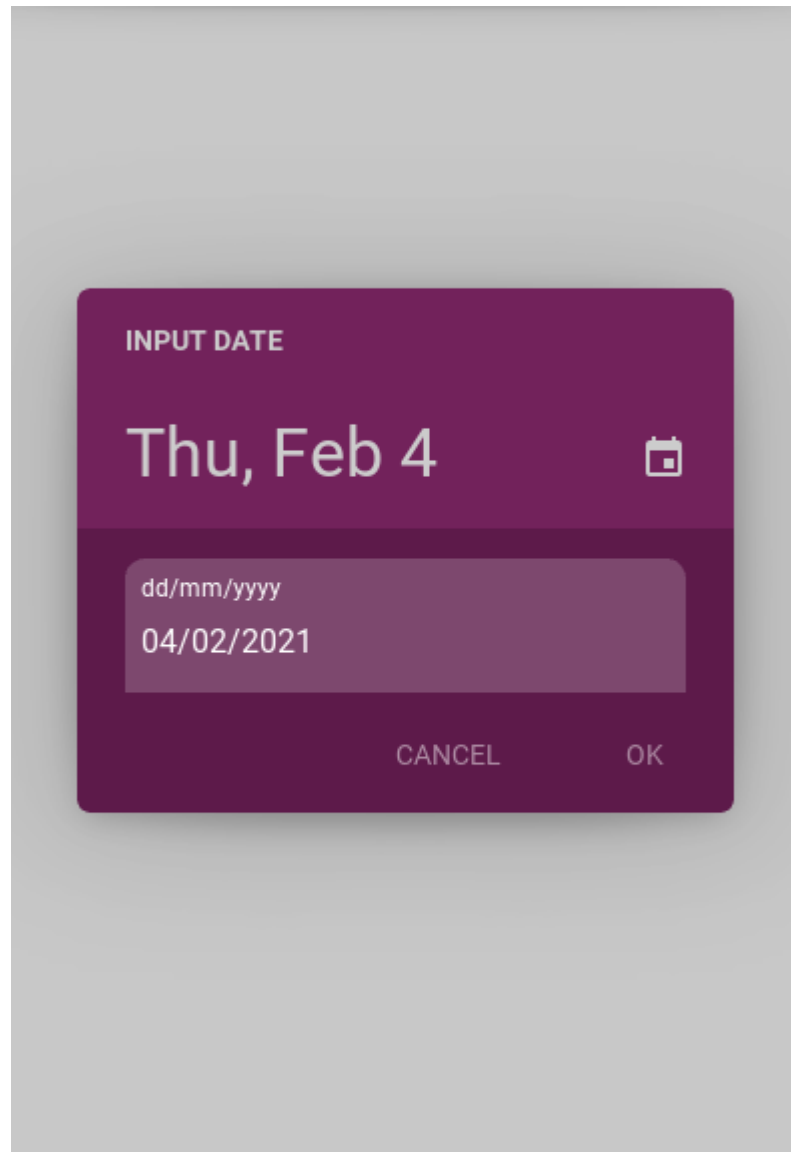


`text_button_color` is an `ColorProperty` and defaults to `None`.

input_field_background_color

Background color of input fields in (r, g, b, a) format.

```
MDDDatePicker(
    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),
)
```



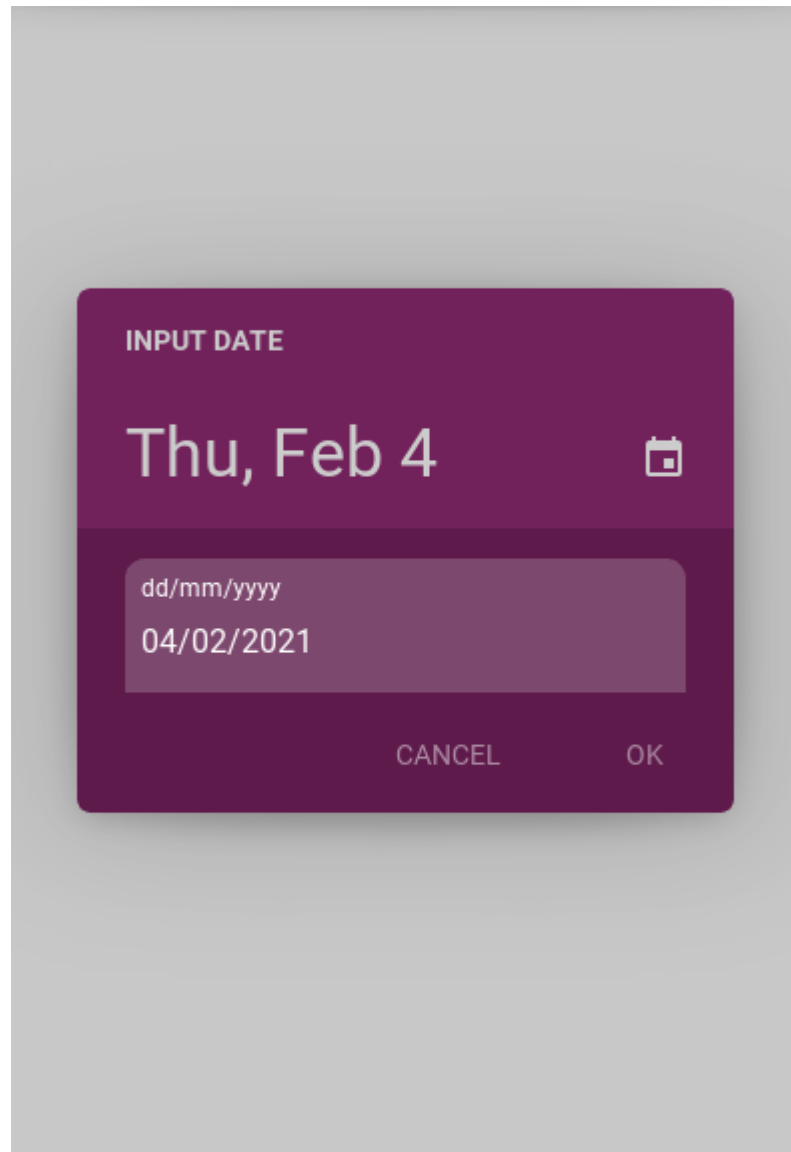
`input_field_background_color` is an `ColorProperty` and defaults to `None`.

input_field_text_color

Text color of input fields in (r, g, b, a) format.

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),
)
```

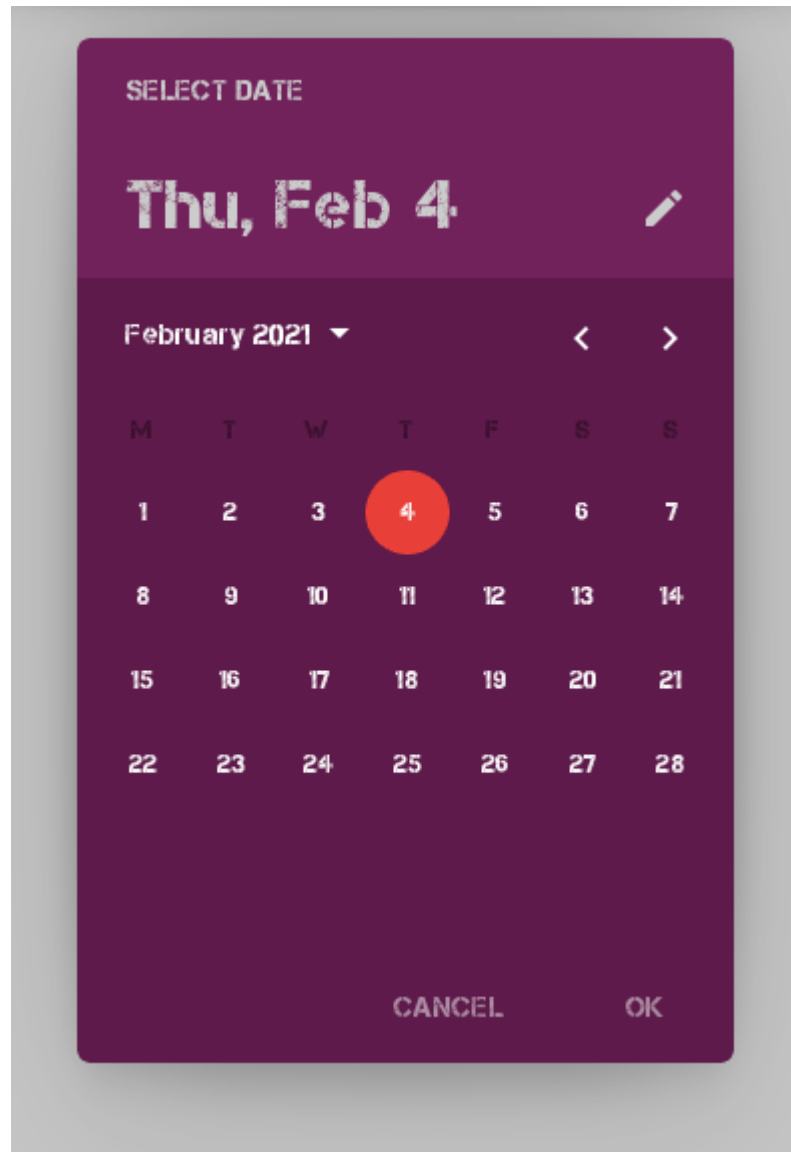


`input_field_text_color` is an `ColorProperty` and defaults to `None`.

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=("#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),
    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.pickers.datepicker.datepicker.DatePickerInputField(kwargs)`**

Implements date input in dd/mm/yyyy format.

`helper_text_mode`

`owner`

`set_error(self)`

Sets a text field to an error state.

input_filter(*self*, *value*: *str*, *boolean*: *bool*)

Filters the input according to the specified mode.

is_numeric(*self*, *value*: *str*)

Returns true if the value of the *value* argument can be converted to an integer, or if the value of the *value* argument is '/'.

get_list_date(*self*)

Returns a list as *[dd, mm, yyyy]* from a text field for entering a date.

class kivymd.uix.pickers.datepicker.datepicker.**MDDatePicker**(*year=None, month=None, day=None, firstweekday=0, **kwargs*)

Base class for MDDatePicker and MDTimePicker classes.

Events

on_save

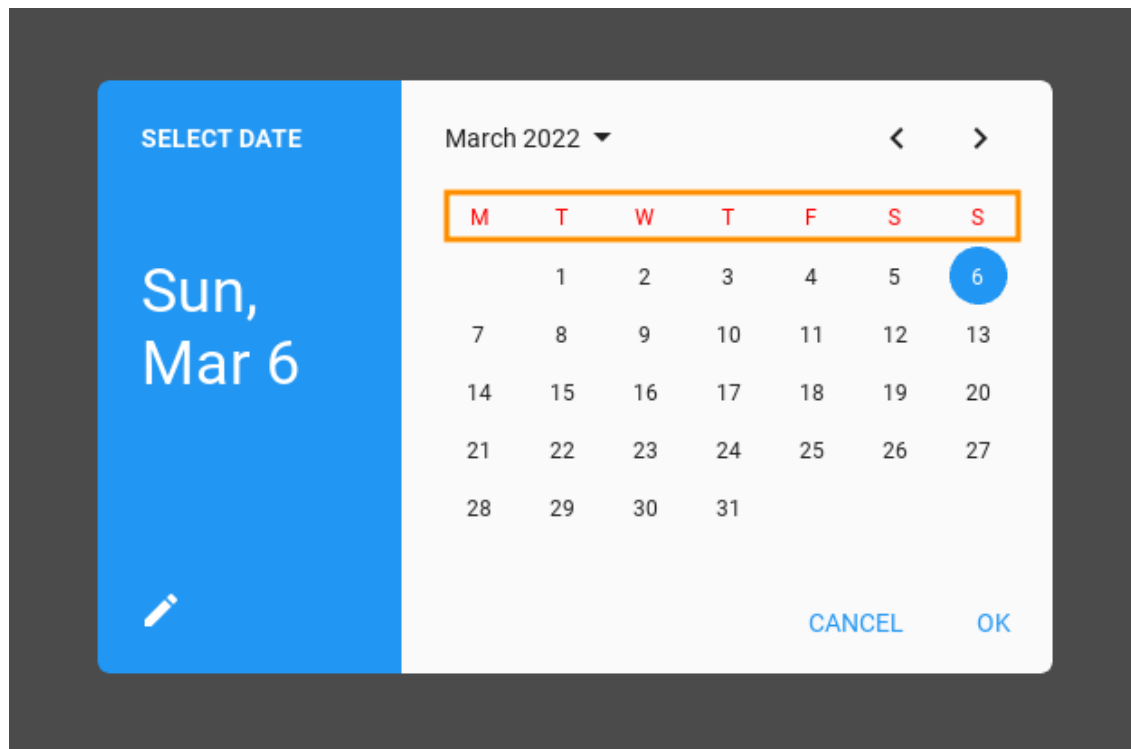
Events called when the “OK” dialog box button is clicked.

on_cancel

Events called when the “CANCEL” dialog box button is clicked.

text_weekday_color

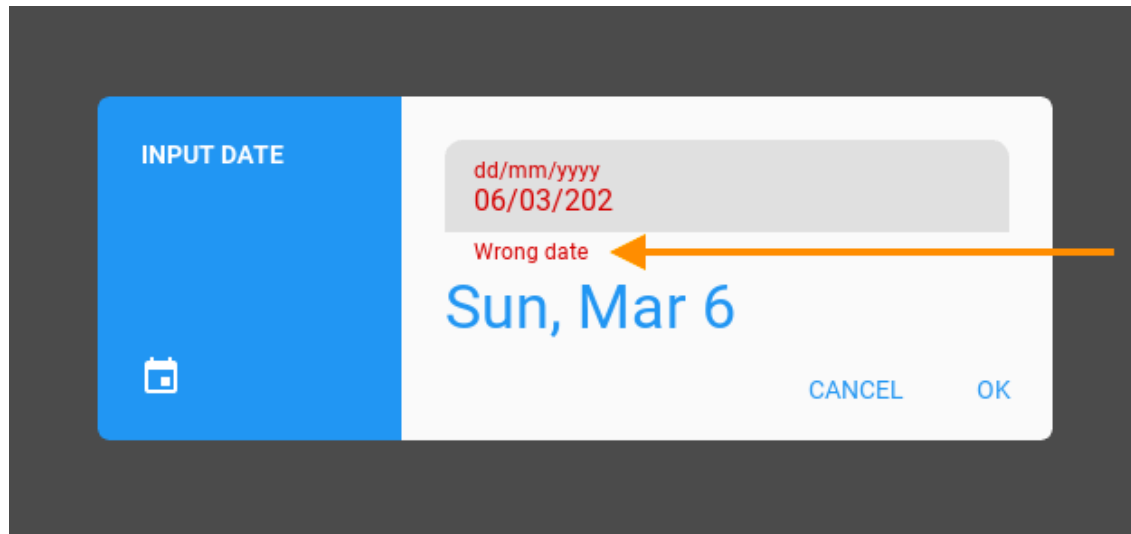
Text color of weekday names in (r, g, b, a) format.



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

helper_text

Helper text when entering an invalid date.



`helper_text` is an `StringProperty` and defaults to `'Wrong date'`.

day

The day of the month to be opened by default. If not specified, the current number will be used.

See [Open date dialog with the specified date](#) for more information.

`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.

See [Open date dialog with the specified date](#) for more information.

`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.

See [Open date dialog with the specified date](#) for more information.

`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'*>.

See [Open date dialog with the specified date](#) for more information.

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'*>.

See [Open date dialog with the specified date](#) for more information.

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'*.

input_field_cls

A class that will implement date input in the format dd/mm/yyyy. See [DatePickerInputField](#) class for more information.

```
class CustomInputField(MDTextField):
    owner = ObjectProperty() # required attribute

    # Required method.
    def set_error(self):
        [...]

    # Required method.
    def get_list_date(self):
        [...]

    # Required method.
    def input_filter(self):
        [...]

    def show_date_picker(self):
        date_dialog = MDDatePicker(input_field_cls=CustomInputField)
```

input_field_cls is an [ObjectProperty](#) and defaults to [DatePickerInputField](#).

sel_year**sel_month****sel_day****on_device_orientation**(self, instance_theme_manager: [ThemeManager](#), orientation_value: *str*)

Called when the device's screen orientation changes.

on_ok_button_pressed(self)

Called when the 'OK' button is pressed to confirm the date entered.

is_date_valaid(self, date: *str*)

Checks the valid of the currently entered date.

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*: *Union[int, float]*)

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*: *str*)

Called when “chevron-left” and “chevron-right” buttons are pressed. Switches the calendar to the previous/next month.

2.3.34 ColorPicker

New in version 1.0.0.

Create, share, and apply color palettes to your UI, as well as measure the accessibility level of any color combination..



Usage

```
from typing import Union

from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.pickers import MDColorPicker

KV = '''
MDScreen:

    MDTopAppBar:
        id: toolbar
        title: "MDTopAppBar"
        pos_hint: {"top": 1}

    MDRaisedButton:
        text: "OPEN PICKER"
        pos_hint: {"center_x": .5, "center_y": .5}
        md_bg_color: toolbar.md_bg_color
        on_release: app.open_color_picker()
```

(continues on next page)

(continued from previous page)

```
'''

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

    def open_color_picker(self):
        color_picker = MDColorPicker(size_hint=(0.45, 0.85))
        color_picker.open()
        color_picker.bind(
            on_select_color=self.on_select_color,
            on_release=self.get_selected_color,
        )

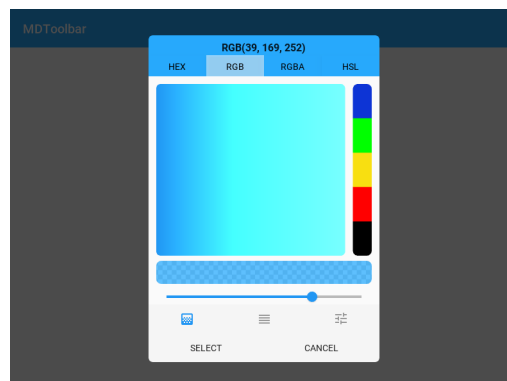
    def update_color(self, color: list) -> None:
        self.root.ids.toolbar.md_bg_color = color

    def get_selected_color(
        self,
        instance_color_picker: MDColorPicker,
        type_color: str,
        selected_color: Union[list, str],
    ):
        '''Return selected color.'''

        print(f"Selected color is {selected_color}")
        self.update_color(selected_color[:-1] + [1])

    def on_select_color(self, instance_gradient_tab, color: list) -> None:
        '''Called when a gradient image is clicked.'''

MyApp().run()
```



API - `kivymd.uix.pickers.colorpicker.colorpicker`

class `kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker(**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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

adjacent_color_constants

A list of values that are used to create the gradient. These values are selected empirically. Each of these values will be added to the selected RGB value, thus creating colors that are close in value.

adjacent_color_constants is an `ListProperty` and defaults to `[0.299, 0.887, 0.411]`.

default_color

Default color value The set color value will be used when you open the dialog.

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

type_color

Type of color. Available options are: 'RGBA', 'HEX', 'RGB'.

type_color is an `OptionProperty` and defaults to 'RGB'.

background_down_button_selected_type_color

Button background for choosing a color type ('RGBA', 'HEX', 'HSL', 'RGB').



background_down_button_selected_type_color is an `ColorProperty` and defaults to `[1, 1, 1, 0.3]`.

radius_color_scale

The radius value for the color scale.



`radius` is an `VariableListProperty` and defaults to `[8, 8, 8, 8]`.

text_button_ok

Color selection button text.

`text_button_ok` is an `StringProperty` and defaults to `'SELECT'`.

text_button_cancel

Cancel button text.

`text_button_cancel` is an `StringProperty` and defaults to `'CANCEL'`.

selected_color

update_color_slider_item_bottom_navigation(*self*, *color*: *list*)

Updates the color of the slider that sets the transparency value of the selected color and the color of bottom navigation items.

update_color_type_buttons(*self*, *color*: *list*)

Updating button colors (display buttons of type of color) to match the selected color.

get_rgb(*self*, *color*: *list*)

Returns an RGB list of values from 0 to 255.

on_background_down_button_selected_type_color(*self*, *instance_color_picker*, *color*: *list*)

on_type_color(*self*, *instance_color_picker*, *type_color*: *str* = "", *interval*: *Union*[*float*, *int*] = 0)

Called when buttons are clicked to set the color type.

on_open(*self*)

Default open event handler.

on_select_color(*self*, *color*: *list*)

Called when a gradient image is clicked.

on_switch_tabs(*self*, *bottom_navigation_instance*, *bottom_navigation_item_instance*, *name_tab*)

Called when switching tabs of bottom navigation.

on_release(*self*, **args*)

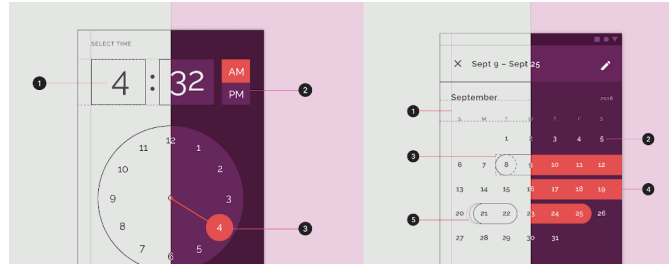
Called when the `SELECT` button is pressed

2.3.35 TimePicker

See also:

Material Design spec, Time picker

Includes time picker.



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.pickers import MDTIMEPicker

KV = '''
MDFloatLayout:

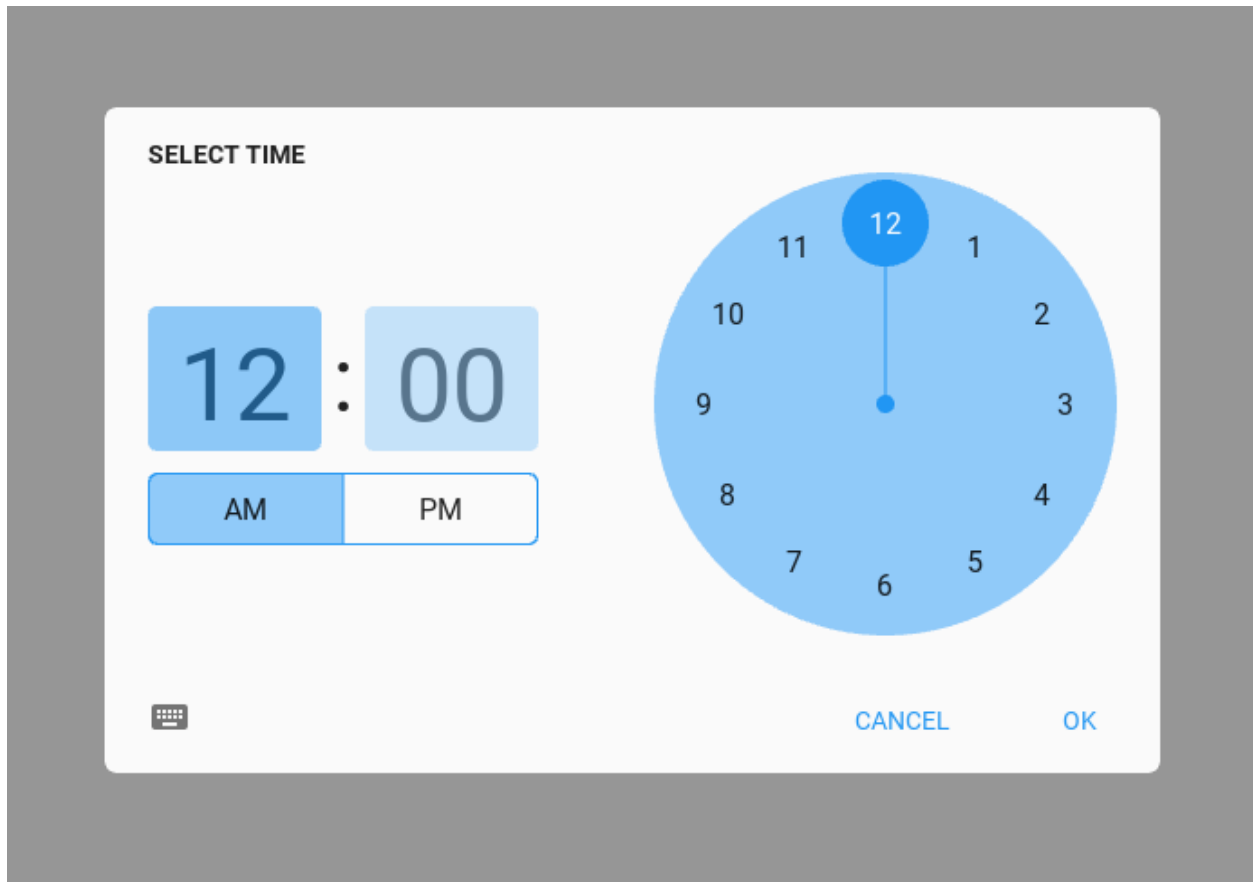
    MDRaisedButton:
        text: "Open time picker"
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_time_picker()
'''

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 = MDTTimePicker()
    time_dialog.set_time(previous_time)
    time_dialog.open()
```

Note: For customization of the `MDTimePicker` class, see the documentation in the `BaseDialogPicker` class.

```
time_dialog = MDTTimePicker(
    primary_color=get_color_from_hex("#72225b"),
    accent_color=get_color_from_hex("#5d1a4a"),
    text_button_color=(1, 1, 1, 1),
)

.. image:: https://github.com/HeaTTheatR/KivyMD-data/raw/master/gallery/kivymddoc/time-
picker-customization.png
    :align: center
```

API - `kivymd.uix.pickers.timepicker.timepicker`

class `kivymd.uix.pickers.timepicker.timepicker.MDTTimePicker(**kwargs)`

Base class for `MDDDatePicker` 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'`.

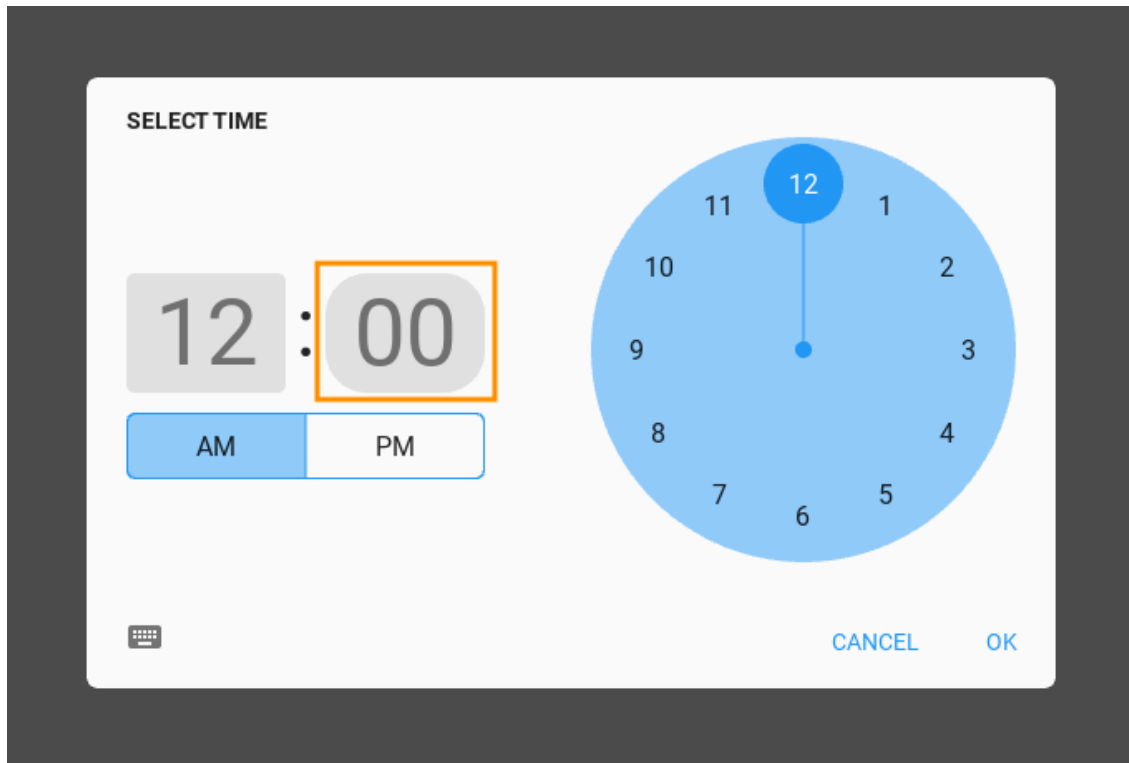
minute

Current minute.

minute is an `StringProperty` and defaults to `0`.

minute_radius

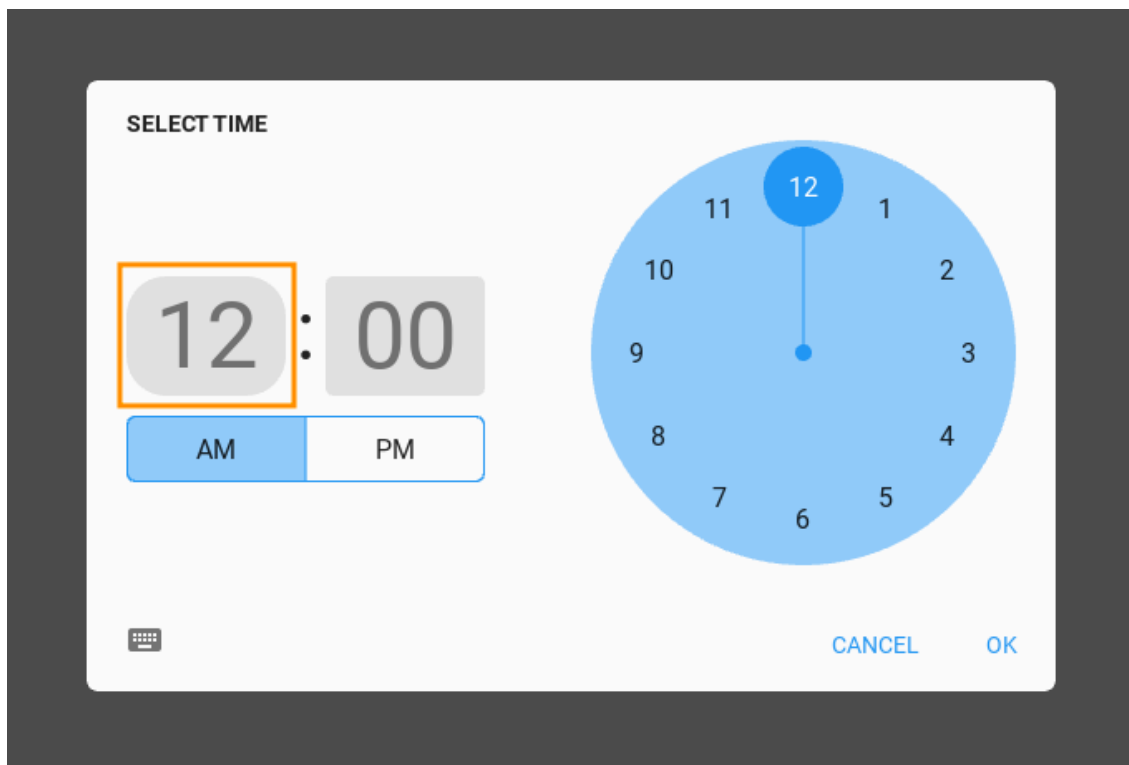
Radius of the minute input field.



minute_radius is an `ListProperty` and defaults to `[dp(5), dp(5), dp(5), dp(5)]`.

hour_radius

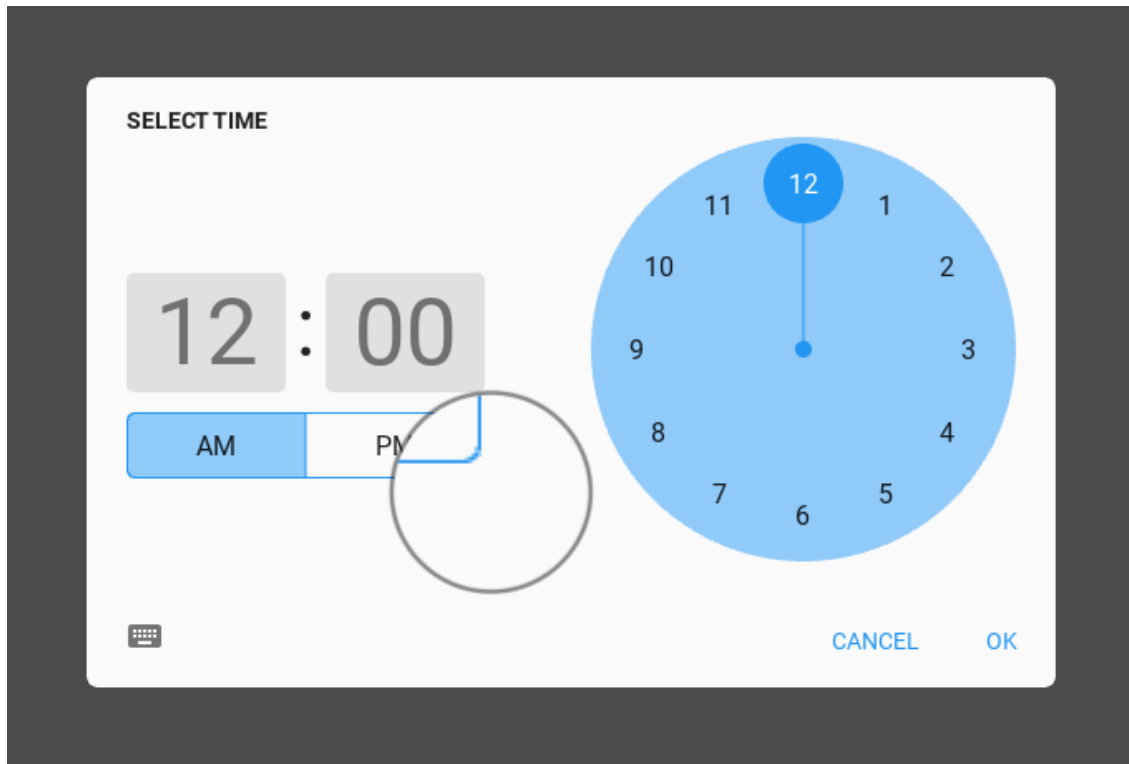
Radius of the hour input field.



hour_radius is an `ListProperty` and defaults to `[dp(5), dp(5), dp(5), dp(5)]`.

am_pm_radius

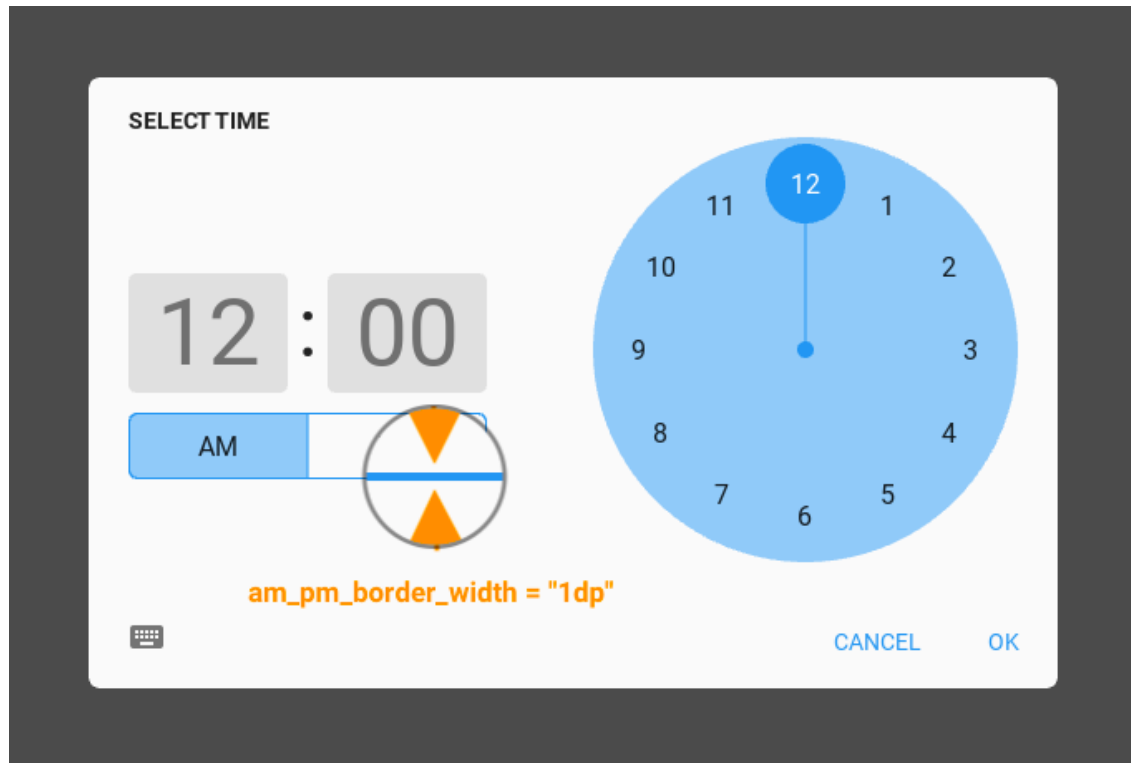
Radius of the AM/PM selector.



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)

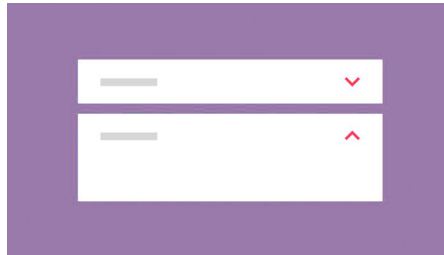
Returns the current state of TimePicker. Can be one of *portrait*, *landscape* or *input*.

2.3.36 ExpansionPanel

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",
        )
    )
)
```


Example

```

import os

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'

MDScrollView:

    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=os.path.join(images_path, "logo", "kivymd-icon-128.png"),
                    content=Content(),
                    panel_cls=MDExpansionPanelThreeLine(
                        text="Text",
                        secondary_text="Secondary text",
                        tertiary_text="Tertiary text",
                    )
                )
            )

```

(continues on next page)

(continued from previous page)

```
)

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.expansionpanel`

```
class kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanelOneLine(*args, **kwargs)
```

Single line panel.

```
class kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanelTwoLine(*args, **kwargs)
```

Two-line panel.

```
class kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanelThreeLine(*args, **kwargs)
```

Three-line panel.

```
class kivymd.uix.expansionpanel.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: *Union[int, float]*)

```
class kivymd.uix.expansionpanel.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_panel: [MDExpansionPanelThreeLine, MDExpansionPanelTwoLine, MDExpansionPanelThreeLine, MDExpansionPanelLabel])

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: MDExpansionChevronRight)

Sets the chevron up.

close_panel(self, instance_expansion_panel, press_current_panel: bool)

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

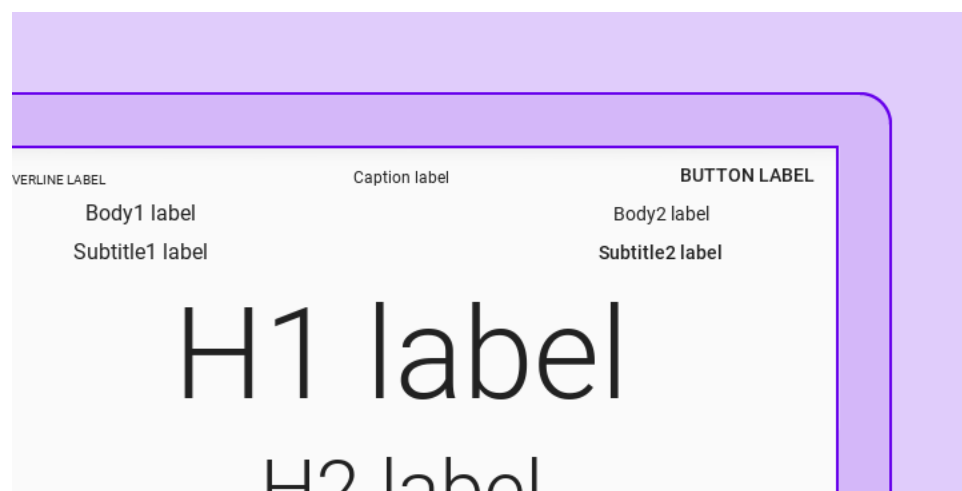
Canvas to add widget's canvas to. Can be 'before', 'after' or None for the default canvas.

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)
```

2.3.37 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:

    MDBoxLayout:
        orientation: "vertical"

        MDTopAppBar:
            title: "MDLabel"

        MDLabel:
            text: "MDLabel"
'''

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

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

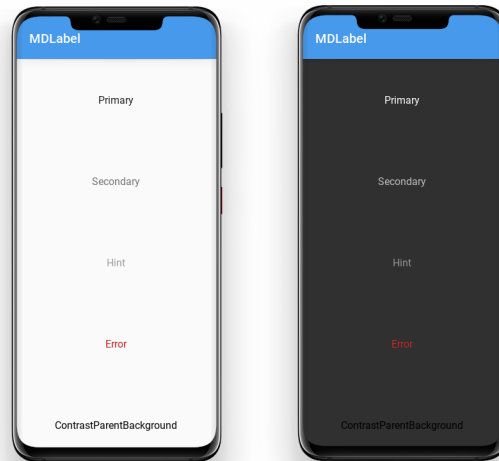
KV = '''
MDScreen:

    MDBoxLayout:
        id: box
        orientation: "vertical"

        MDTopAppBar:
            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",
                    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
```

Custom color

`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:

    MDBoxLayout:
        orientation: "vertical"

        MDTopAppBar:
            title: "MDLabel"

        ScrollView:

            MDList:
                id: box
'''
```

(continues on next page)

(continued from previous page)

```

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:
    icon: "gmail"
    pos_hint: {"center_x": .5, "center_y": .5}

```



MDIcon with badge icon

```
MDIcon:
    icon: "gmail"
    badge_icon: "numeric-10"
    pos_hint: {"center_x": .5, "center_y": .5}
```



API - kivymd.uix.label.label

class kivymd.uix.label.label.MDLabel(**kwargs)

Implements the creation and addition of child widgets as declarative programming style.

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 (r, g, b, a) format.

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

parent_background

can_capitalize

check_font_styles(self, interval: Union[int, float] = 0)

update_font_style(self, instance_label, font_style: str)

on_theme_text_color(self, instance_label, theme_text_color: str)

on_text_color(self, instance_label, color: list)

on_opposite_colors(*self*, **args*)

class kivymd.uix.label.label.MDIcon(***kwargs*)

Float layout class. For more information, see in the [FloatLayout](#) class documentation.

icon

Label icon name.

icon is an [StringProperty](#) and defaults to *'android'*.

badge_icon

Label badge icon name.

New in version 1.0.0.

badge_icon is an [StringProperty](#) and defaults to *''*.

badge_icon_color

Badge icon color in (r, g, b, a) format.

New in version 1.0.0.

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

badge_bg_color

Badge icon background color in (r, g, b, a) format.

New in version 1.0.0.

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

badge_font_size

Badge font size.

New in version 1.0.0.

badge_font_size is an [NumericProperty](#) and defaults to *0*.

source

Path to icon.

source is an [StringProperty](#) and defaults to *None*.

2.3.38 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 simple, gold-colored frame. It showed a lady fitted out with a fur hat and fur boa who sat upright, raising a heavy fur muff that covered the whole of her lower arm towards the camera.

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

Undo

Redo

Cut

Copy

Paste



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 `RecyclerView` class.

```

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

from kivymd.app import MDApp
from kivymd.ui.boxlayout import MDBoxLayout
from kivymd.ui.list import IRightBodyTouch, OneLineAvatarIconListItem
from kivymd.ui.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, MDBoxLayout):
    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

```

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 MDBoxLayout

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

    MDBoxLayout:
        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.'''

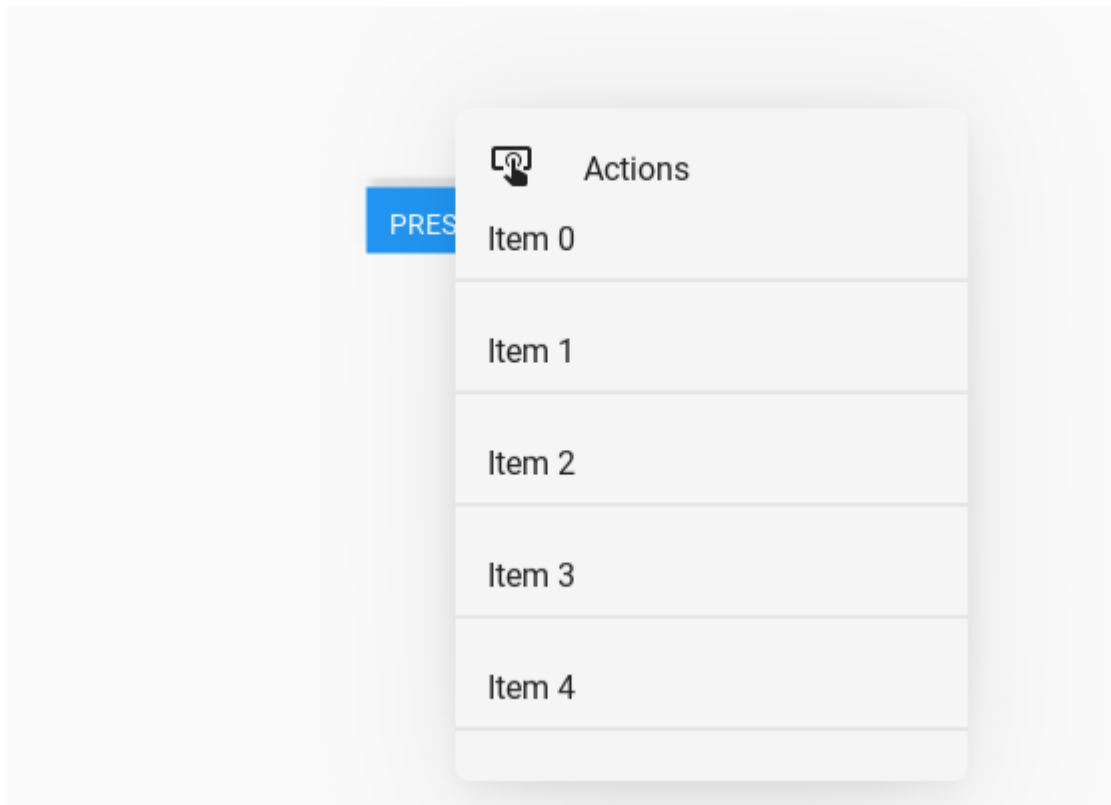
class Test(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = Builder.load_string(KV)
        menu_items = [
            {
                "text": f"Item {i}",
                "viewclass": "OneLineListItem",
                "height": dp(56),
                "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 MDTopAppBar

The `MDDropdownMenu` works well with the standard `MDTopAppBar`. Since the buttons on the Toolbar are created by the `MDTopAppBar` 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 `MDNavigationDrawer`.

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

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

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

    MDTopAppBar:
        title: "MDTopAppBar"
        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

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.menu`

```
class kivymd.uix.menu.menu.MDDropdownMenu(**kwargs)
```

Events

on_release

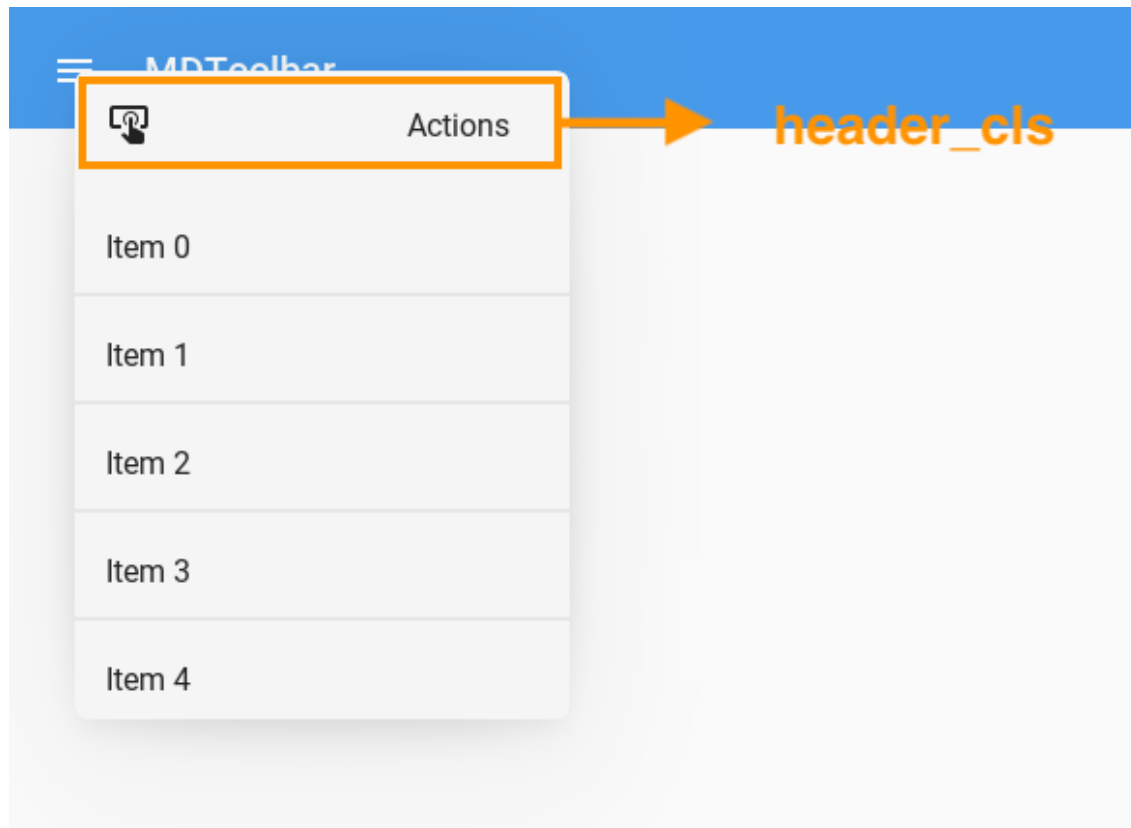
The method that will be called when you click menu items.

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.

See [Header](#) for more information.

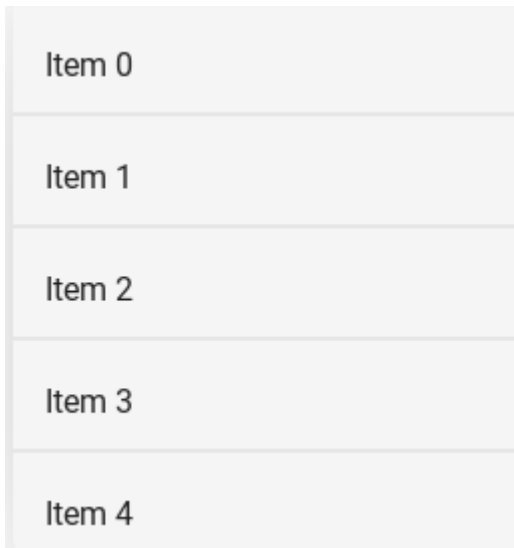


`header_cls` is a `ObjectProperty` and defaults to `None`.

items

See `data`.

```
items = [
    {
        "viewclass": "OneLineListItem",
        "height": dp(56),
        "text": f"Item {i}",
    }
    for i in range(5)
]
self.menu = MDDropdownMenu(
    items=items,
    ...,
)
```



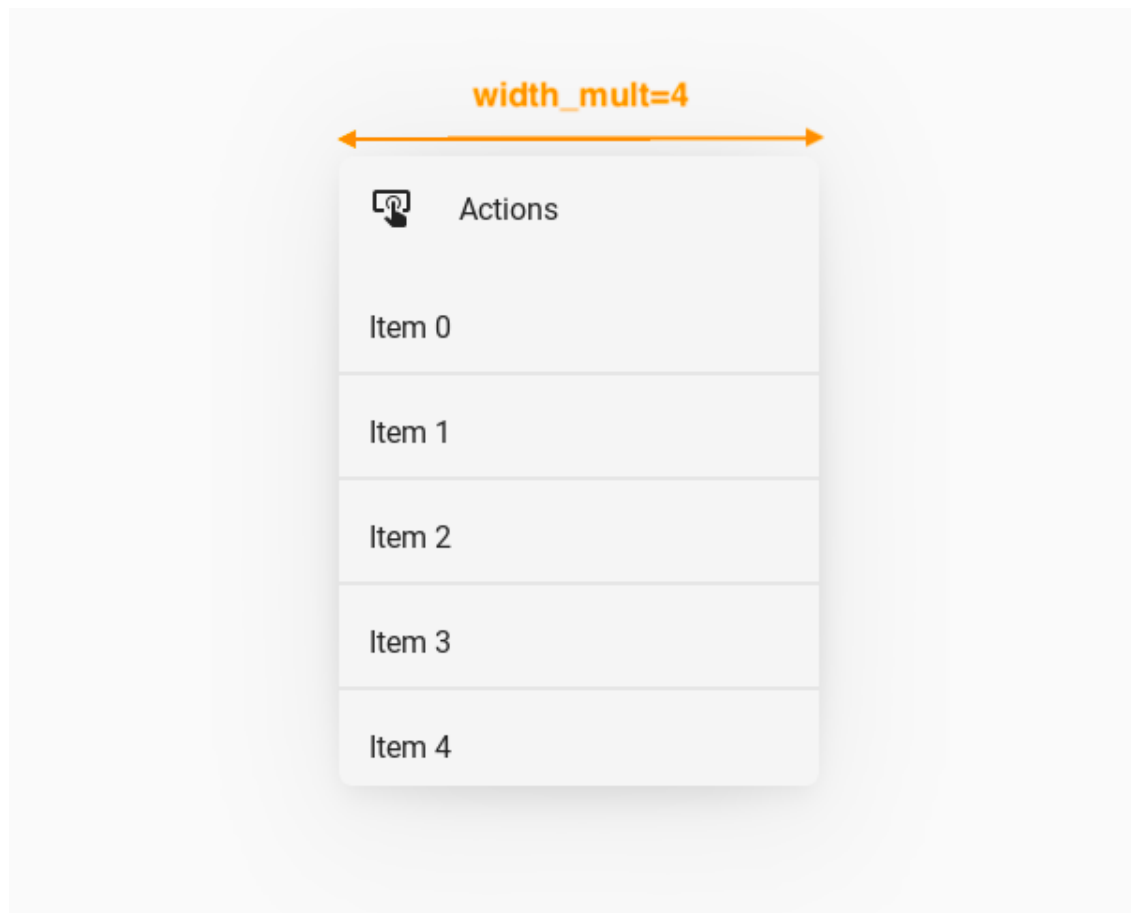
`items` is a `ListProperty` and defaults to `[]`.

width_mult

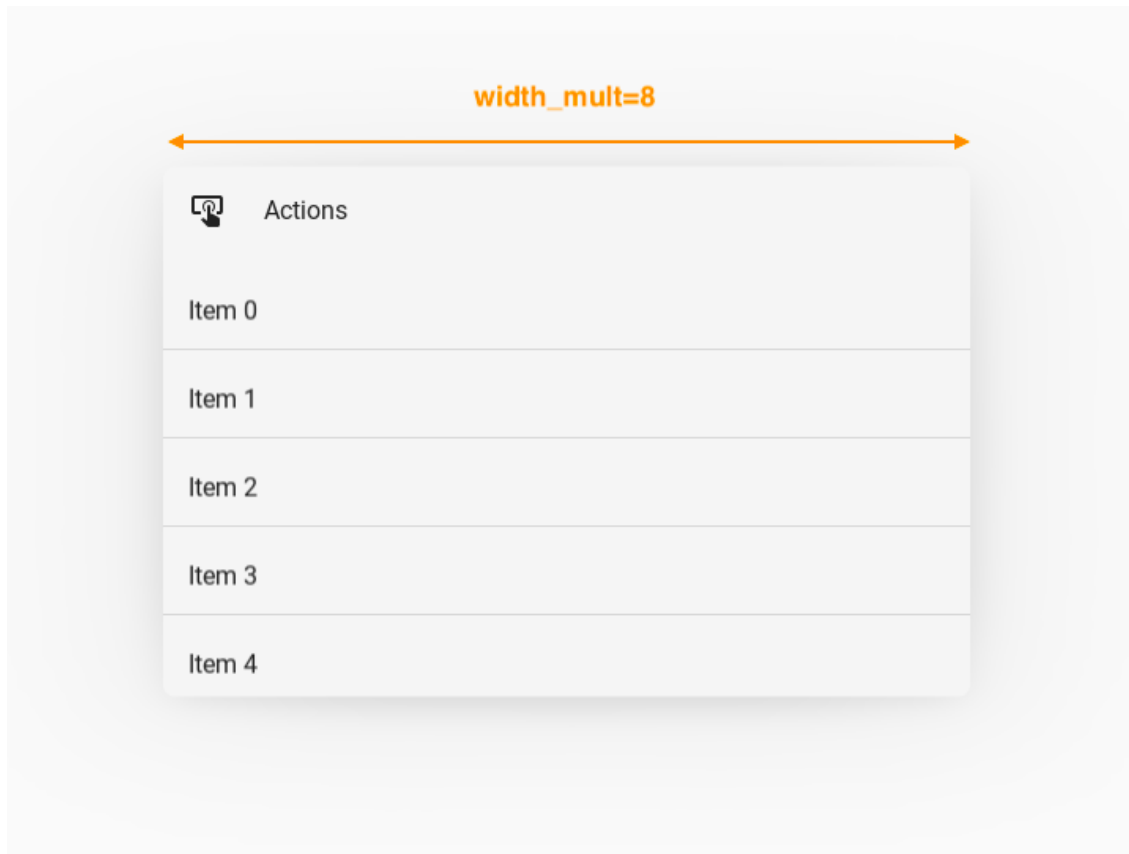
This number multiplied by the standard increment ('56dp' on mobile, '64dp' on desktop), determines the width of the menu items.

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

```
self.menu = MDDropdownMenu(
    width_mult=4,
    ...,
)
```



```
self.menu = MDDropdownMenu(  
    width_mult=8,  
    ...,  
)
```



`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.

```
self.menu = MDDropdownMenu(  
    max_height=dp(112),  
    ...,  
)
```



```
self.menu = MDDropdownMenu(  
    max_height=dp(224),  
    ...,  
)
```

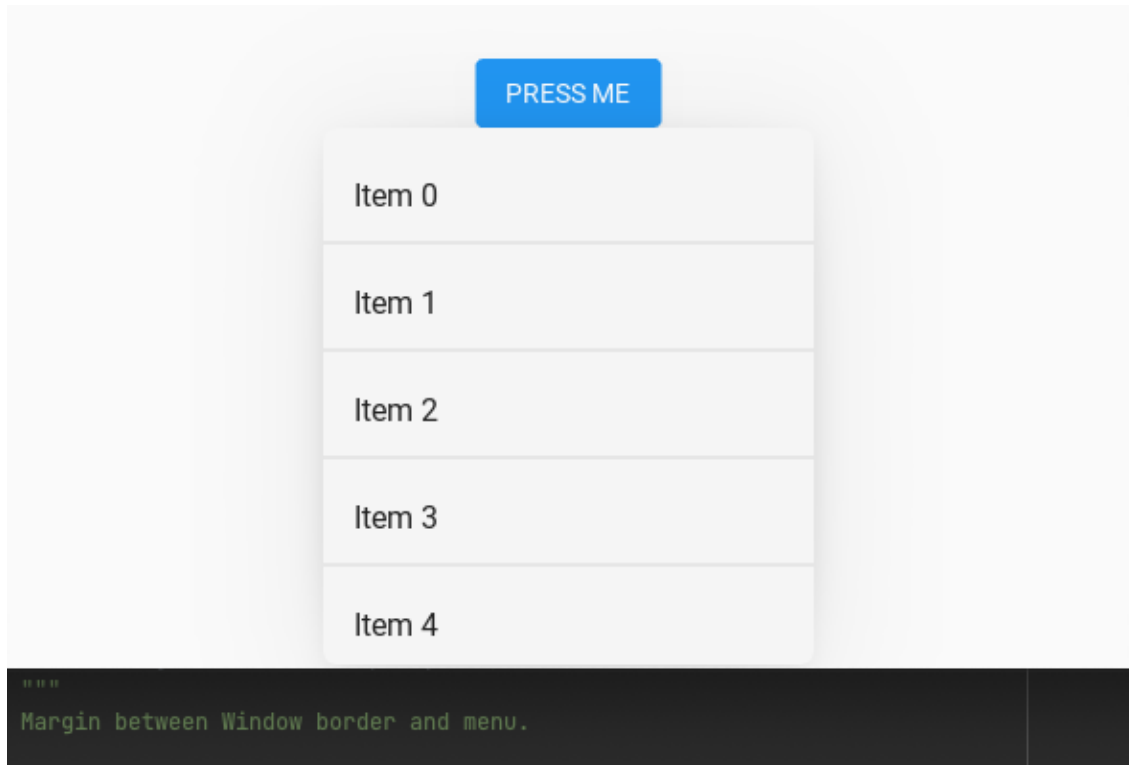


max_height is a `NumericProperty` and defaults to `0`.

border_margin

Margin between Window border and menu.


```
self.menu = MDDropdownMenu(  
    border_margin=dp(4),  
    ...,  
)
```



```
self.menu = MDDropdownMenu(  
    border_margin=dp(24),  
    ...,  
)
```



`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'`.

```
self.menu = MDDropdownMenu(
    ver_growth="up",
    ...,
)
```

```
self.menu = MDDropdownMenu(
    ver_growth="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'`.

```
self.menu = MDDropdownMenu(
    hor_growth="left",
    ...,
)
```

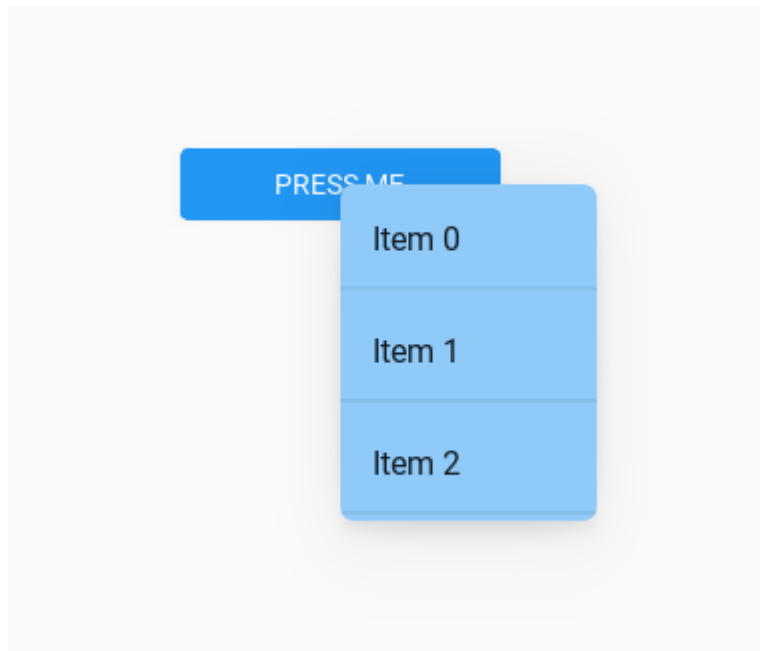
```
self.menu = MDDropdownMenu(
    hor_growth="right",
    ...,
)
```

hor_growth is a *OptionProperty* and defaults to *None*.

background_color

Color of the background of the menu.

```
self.menu = MDDropdownMenu(
    background_color=self.theme_cls.primary_light,
    ...,
)
```



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

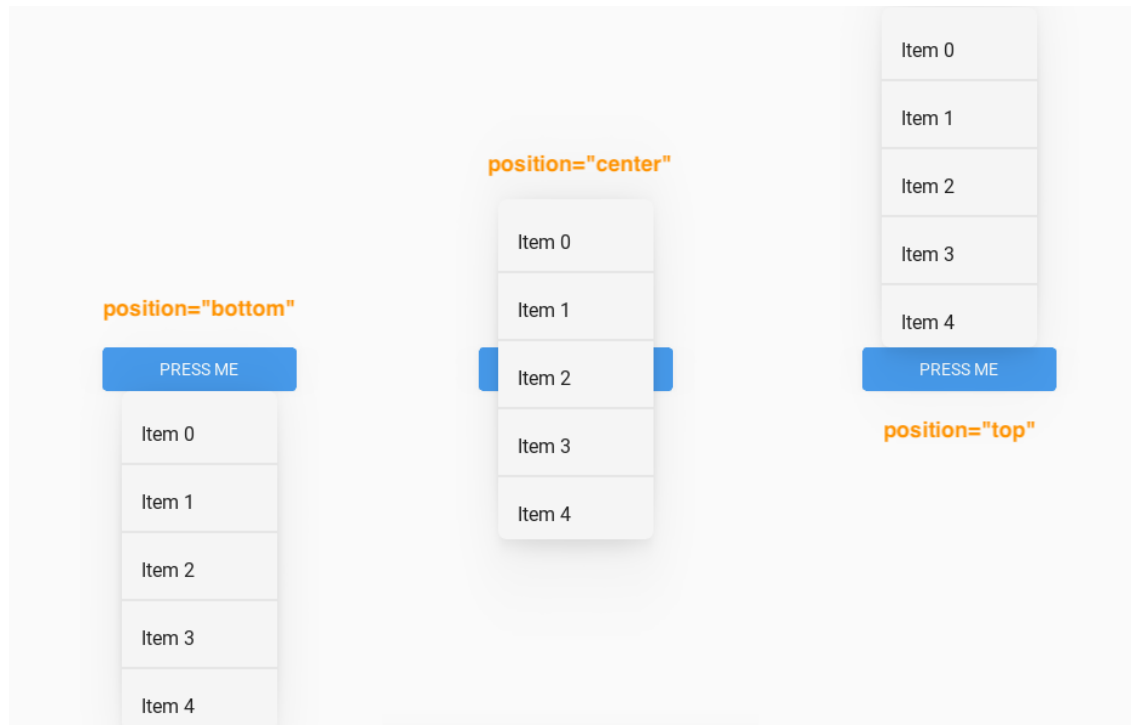
The widget object that calls the menu window.

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

position

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

See [Position](#) for more information.

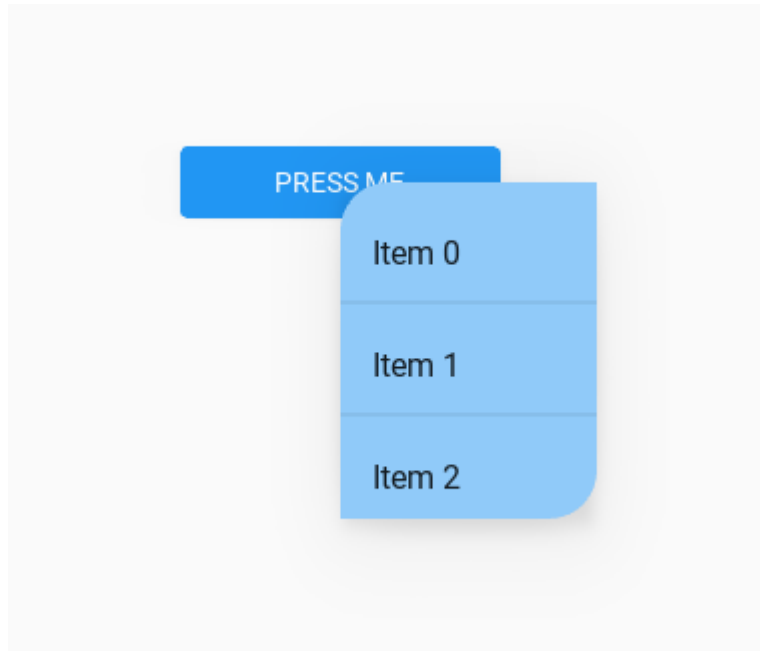


position is a [OptionProperty](#) and defaults to *'auto'*.

radius

Menu radius.

```
self.menu = MDDropdownMenu(  
    radius=[24, 0, 24, 0],  
    ...,  
)
```



`radius` is a `VariableListProperty` and defaults to `[dp(7)]`.

elevation

Elevation value of menu dialog.

New in version 1.0.0.

```
self.menu = MDDropdownMenu(
    elevation=16,
    ...,
)
```



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

check_position_caller(*self*, *instance_window*: `WindowSDL`, *width*: `int`, *height*: `int`)

Called when the application root window is resized.

set_menu_properties(*self*, *interval*: `Union[int, float]` = 0)

Sets the size and position for the menu window.

ajust_radius(*self*, *interval*: *Union[int, float]*)

Adjusts the radius of the first and last items in the menu list according to the radius that is set for the menu.

adjust_position(*self*)

Returns value 'auto' for the menu position if the menu position is out of screen.

open(*self*)

Animate the opening of a menu window.

on_header_cls(*self*, *instance_dropdown_menu*, *instance_user_menu_header*)

Called when a value is set to the [header_cls](#) parameter.

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*, **args*)

Closes the menu.

2.3.39 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 = '''
MDScreen:

    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],
```

```
MDSpinner(
    size_hint=(None, None),
    size=(dp(46), dp(46)),
    pos_hint={'center_x': .5, 'center_y': .5},
    active=True,
    palette=[
```

(continues on next page)

(continued from previous page)

```
[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.spinner

class kivymd.uix.spinner.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_spinner, palette_list: *list*)**on_active**(*self*, instance_spinner, active_value: *bool*)**on_determinate_complete**(*self*, *args)

The event is called at the end of the spinner loop in the *determinate = True* mode.

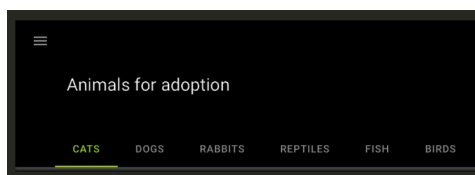
check_determinate(*self*, interval: *Union[float, int] = 0*)

2.3.40 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.'''
```

```
<Tab>

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

Declarative KV and imperative python styles

```
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"

    MDTopAppBar:
        title: "Example Tabs"
```

(continues on next page)

(continued from previous page)

```

MDTabs:
    id: tabs
    on_tab_switch: app.on_tab_switch(*args)

<Tab>

MDIconButton:
    id: icon
    icon: root.icon
    icon_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):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        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;
        '''

        count_icon = instance_tab.icon # get the tab icon
        print(f"Welcome to {count_icon}' tab")

Example().run()

```

Declarative python styles

```

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDIconButton
from kivymd.uix.tab import MDTabsBase, MDTabs
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.icon_definitions import md_icons
from kivymd.uix.toolbar import MDTopAppBar

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

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

    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDBoxLayout(
                MDTopAppBar(title="Example Tabs"),
                MDTabs(id="tabs",
                    orientation="vertical",
                )
            )

    def on_start(self):
        self.root.ids.tabs.bind(on_tab_switch=self.on_tab_switch)

        for tab_name in self.icons:
            self.root.ids.tabs.add_widget(
                Tab(
                    MDIconButton(
                        icon=tab_name,
                        icon_size="48sp",
                        pos_hint={"center_x": .5, "center_y": .5},
                    ),
                    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;

```

(continues on next page)

(continued from previous page)

```
'''
    count_icon = instance_tab.icon # get the tab icon
    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`.

Declarative KV and imperative python styles

```
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"

    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.'''
```

(continues on next page)

(continued from previous page)

```

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        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.

        :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()

```

Declarative python style

```

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.label import MDLabel
from kivymd.uix.tab import MDTabsBase, MDTabs
from kivymd.uix.toolbar import MDTopAppBar

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

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDBoxLayout(
                MDTopAppBar(title="Example Tabs"),
                MDTabs(id="tabs",
                    orientation="vertical",
                )
            )
        )

```

(continues on next page)

(continued from previous page)

```

def on_start(self):
    self.root.ids.tabs.bind(on_tab_switch=self.on_tab_switch)
    for i in range(20):
        self.root.ids.tabs.add_widget(
            Tab(
                MDLabel(id="label", text="Tab 0", halign="center"),
                title=f"Tab {i}",
            )
        )

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;
    '''

    instance_tab.ids.label.text = tab_text

```

Example().run()

Example with tab icon and text

Declarative KV and imperative python styles

```

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"

    MDTopAppBar:
        title: "Example Tabs"

    MDTabs:
        id: tabs
'''

```

(continues on next page)

(continued from previous page)

```

class Tab(MDFloatLayout, MDTabsBase):
    pass

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        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()

```

Declarative python style

```

from kivymd.app import MDApp
from kivymd.ui.boxlayout import MDBoxLayout
from kivymd.ui.tab import MDTabsBase, MDTabs
from kivymd.ui.floatlayout import MDFloatLayout
from kivymd.icon_definitions import md_icons
from kivymd.ui.toolbar import MDTopAppBar

class Tab(MDFloatLayout, MDTabsBase):
    pass

class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDBoxLayout(
                MDTopAppBar(title="Example Tabs"),
                MDTabs(id="tabs",
                    orientation="vertical",
                )
            )

    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()

```


Dynamic tab management

Declarative KV and imperative python styles

```

from kivy.lang import Builder

from kivymd.uix.scrollview import MDScrollView
from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsBase

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

    MDTopAppBar:
        title: "Example Tabs"

    MDTabs:
        id: tabs

<Tab>

    MDList:

        MDBoxLayout:
            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(MDScrollView, MDTabsBase):
    '''Class implementing content for a tab.'''

class Example(MDApp):
    index = 0

    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return Builder.load_string(KV)

```

(continues on next page)

(continued from previous page)

```

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(title=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()

Declarative python style

```

from kivymd.uix.button import MDFlatButton
from kivymd.uix.list import MDList
from kivymd.uix.scrollview import MDScrollView
from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.tab import MDTabsBase, MDTabs
from kivymd.uix.toolbar import MDTopAppBar

class Tab(MDScrollView, MDTabsBase):
    '''Class implementing content for a tab.'''

class Example(MDApp):
    index = 0

    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDBoxLayout(
                MDTopAppBar(title="Example Tabs"),
                MDTabs(id="tabs",
                    orientation="vertical",
                )
            )
        )

```

(continues on next page)

(continued from previous page)

```

def on_start(self):
    self.add_tab()

def get_tab_list(self, *args):
    '''Prints a list of tab objects.'''

    print(self.root.ids.tabs.get_tab_list())

def add_tab(self, *args):
    self.index += 1
    self.root.ids.tabs.add_widget(
        Tab(
            MDList(
                MDBoxLayout(
                    MDFlatButton(
                        text="ADD TAB",
                        on_release=self.add_tab,
                    ),
                    MDFlatButton(
                        text="REMOVE LAST TAB",
                        on_release=self.remove_tab,
                    ),
                    MDFlatButton(
                        text="GET TAB LIST",
                        on_release=self.get_tab_list,
                    ),
                ),
                adaptive_height=True,
            ),
        ),
        title=f"{self.index} tab",
    )

def remove_tab(self, *args):
    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:

Declarative KV and imperative python styles

```
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"

    MDTopAppBar:
        title: "Example Tabs"

    MDTabs:
        id: tabs
        on_ref_press: app.on_ref_press(*args)

<Tab>

    MDIconButton:
        id: icon
        icon: app.icons[0]
        icon_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):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return Builder.load_string(KV)

    def on_start(self):
        for name_tab in self.icons:
            self.root.ids.tabs.add_widget(
                Tab(
                    title=f"[ref={name_tab}][font={fonts[-1]}['fn_regular']] {md_icons[
```

(continues on next page)

(continued from previous page)

```

↪ 'close']]
```

```

Example().run()
```

Declarative python style

```

from kivymd.app import MDApp
from kivymd.ui.boxlayout import MDBoxLayout
from kivymd.ui.button import MDIconButton
from kivymd.ui.floatlayout import MDFloatLayout
from kivymd.font_definitions import fonts
from kivymd.ui.tab import MDTabsBase, MDTabs
from kivymd.icon_definitions import md_icons
from kivymd.ui.toolbar import MDTopAppBar

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

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

    def build(self):
```

(continues on next page)

(continued from previous page)

```

self.theme_cls.theme_style = "Dark"
self.theme_cls.primary_palette = "Orange"
return (
    MDBoxLayout(
        MDTopAppBar(title="Example Tabs"),
        MDTabs(id="tabs",
            orientation="vertical",
        )
    )
)

def on_start(self):
    self.root.ids.tabs.bind(on_ref_press=self.on_ref_press)
    for name_tab in self.icons:
        self.root.ids.tabs.add_widget(
            Tab(
                MDIconButton(
                    icon=self.icons[0],
                    icon_size="48sp",
                    pos_hint={"center_x": .5, "center_y": .5}
                ),
                title=(
                    f"[ref={name_tab}][font={fonts[-1]['fn_regular'}}]"
                    f"{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:
        if instance_tab.title == instance_tab_label.text:
            instance_tabs.remove_widget(instance_tab_label)
            break

```

(continues on next page)

(continued from previous page)

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

Switching the tab by name

Declarative KV and imperative python styles

```
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"

    MDTopAppBar:
        title: "Example Tabs"

    MDTabs:
        id: tabs

<Tab>

    MDBoxLayout:
        orientation: "vertical"
        pos_hint: {"center_x": .5, "center_y": .5}
        adaptive_size: True
        spacing: dp(48)

        MDIconButton:
            id: icon
            icon: "arrow-right"
            icon_size: "48sp"
            on_release: app.switch_tab_by_name()

        MDIconButton:
            id: icon2
            icon: "page-next"
            icon_size: "48sp"
            on_release: app.switch_tab_by_object()
'''

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

(continues on next page)

(continued from previous page)

```

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

    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        self.iter_list_names = iter(list(self.icons))
        return Builder.load_string(KV)

    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, next 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()

    def switch_tab_by_name(self):
        '''Switching the tab by name.'''

        try:
            x = next(self.iter_list_names)
            print(f"Switch slide by name, next 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()

Example().run()

```

Declarative python style

```

from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.icon_definitions import md_icons
from kivymd.ui.boxlayout import MDBoxLayout
from kivymd.ui.button import MDIconButton
from kivymd.ui.floatlayout import MDFloatLayout
from kivymd.ui.tab import MDTabsBase, MDTabs
from kivymd.ui.toolbar import MDTopAppBar

```

(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.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        self.iter_list_names = iter(list(self.icons))
        return (
            MDBoxLayout(
                MDTopAppBar(title="Example Tabs"),
                MDTabs(id="tabs",
                    orientation="vertical",
                )
            )

    def on_start(self):
        for name_tab in list(self.icons):
            self.root.ids.tabs.add_widget(
                Tab(
                    MDBoxLayout(
                        MDIconButton(
                            id="icon",
                            icon="arrow-right",
                            icon_size="48sp",
                            on_release=self.switch_tab_by_name,
                        ),
                        MDIconButton(
                            id="icon2",
                            icon="arrow-left",
                            icon_size="48sp",
                            on_release=self.switch_tab_by_object,
                        ),
                    ),
                    orientation="vertical",
                    pos_hint={"center_x": .5, "center_y": .5},
                    adaptive_size=True,
                    spacing=dp(48),
                ),
                tab_label_text=name_tab,
            )

        self.iter_list_objects = iter(list(self.root.ids.tabs.get_tab_list()))

    def switch_tab_by_object(self, *args):
        try:
            x = next(self.iter_list_objects)

```

(continues on next page)

(continued from previous page)

```

        print(f"Switch slide by object, next 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()

    def switch_tab_by_name(self, *args):
        '''Switching the tab by name.'''

        try:
            x = next(self.iter_list_names)
            print(f"Switch slide by name, next 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()

```

Example().run()

API - kivymd.uix.tab.tab

class kivymd.uix.tab.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 wich 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`.

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*, *instance_user_tab*, *text_tab*: *str*)

class kivymd.uix.tab.tab.MDTabs(*args, **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.

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 it's content width, and wrap any text if the widget is wider than *"360dp"*.

If *False*, the tab won't update to it's 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_theme_manager: `ThemeManager`, name_theme_style_name_palette: *str*)

Called when the app's color scheme or style has changed (dark theme/light theme).

switch_tab(self, name_tab: `Union[MDTabsLabel, str]`, search_by='text')

This method 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 `MDTabsLabel` objects.

get_slides(self)

Returns a list of user tab objects.

get_current_tab(self)

Returns current tab object.

New in version 1.0.0.

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)
```

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*, *instance_tabs_carousel*, *index: int*)

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*, *instance_tab*, *size: list*)

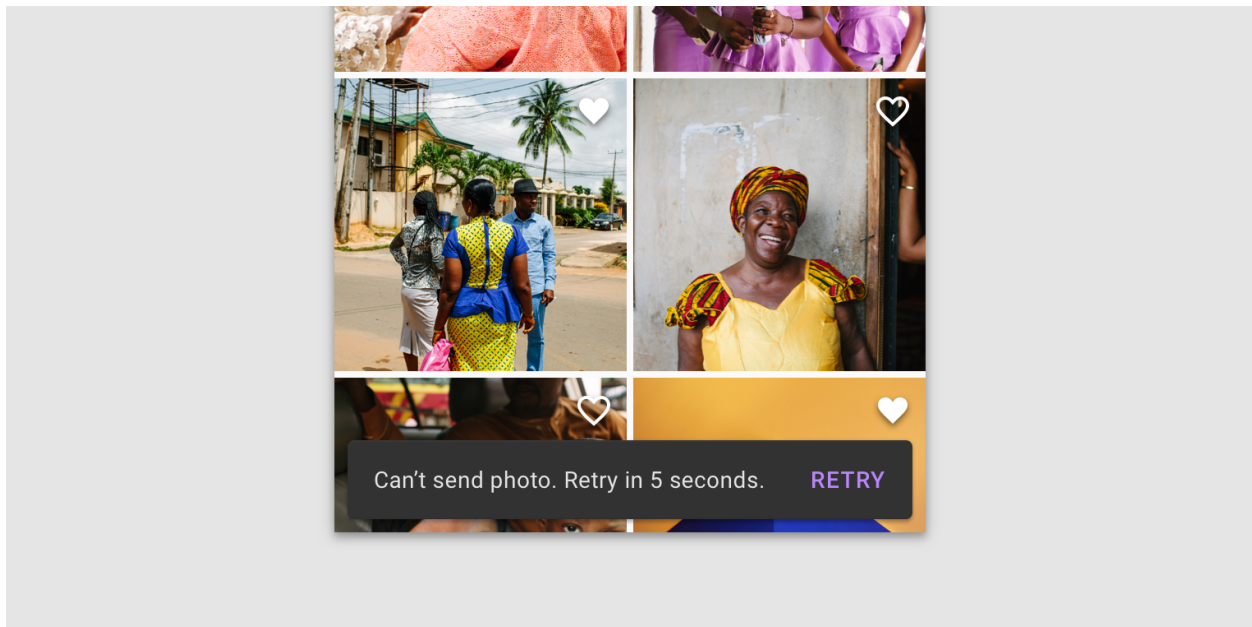
Called when the application screen is resized.

2.3.41 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

MDScreen:

    MDRaisedButton:
        text: "Create simple snackbar"
        on_release: Snackbar(text="This is a snackbar!").open()
        pos_hint: {"center_x": .5, "center_y": .5}
...

class Test(MDApp):
    def build(self):
```

(continues on next page)

(continued from previous page)

```
        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()
```




Custom text color

```
Snackbar(
    text="[color=#ddbb34]This is a snackbar![/color]",
    snackbar_y="10dp",
    snackbar_x="10dp",
    size_hint_x=.7
).open()
```



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()
```



UPDATE

CANCEL

Using a button with custom color

```
Snackbar(
    ...
    bg_color=(0, 0, 1, 1),
).open()
```



UPDATE

CANCEL

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 = '''
MDScreen:

    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)
            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: 'ffffff'
        shorten: True
        shorten_from: 'right'
        pos_hint: {'center_y': .5}

MDScreen:

    MDRaisedButton:
        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)

```

(continues on next page)

(continued from previous page)

```
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.snackbar`

```
class kivymd.uix.snackbar.snackbar.BaseSnackbar(**kwargs)
```

Events

`on_open`

Called when a dialog is opened.

`on_dismiss`

When the front layer rises.

Abstract base class for all Snackbars. This class handles sizing, positioning, shape and events for Snackbars

All Snackbars will be made off of this *BaseSnackbar*.

BaseSnackbar will always try to fill the remainder of the screen with your Snackbar.

To make your Snackbar dynamic and symetric with `snackbar_x`.

Set `size_hint_x` like below:

```
size_hint_x = (
    Window.width - (snackbar_x * 2)
) / Window.width
```

`duration`

The amount of time that the snackbar will stay on screen for.

`duration` is a `NumericProperty` and defaults to 3.

`auto_dismiss`

Whether to use automatic closing of the snackbar or not.

`auto_dismiss` is a `BooleanProperty` and defaults to 'True'.

bg_color

Snackbar background.

bg_color is a [ColorProperty](#) and defaults to *None*.

buttons

Snackbar buttons.

buttons is a [ListProperty](#) and defaults to *[]*

radius

Snackbar radius.

radius is a [ListProperty](#) and defaults to *[5, 5, 5, 5]*

snackbar_animation_dir

Snackbar animation direction.

Available options are: “*Top*”, “*Bottom*”, “*Left*”, “*Right*”

snackbar_animation_dir is an [OptionProperty](#) and defaults to *‘Bottom’*.

snackbar_x

The snackbar x position in the screen

snackbar_x is a [NumericProperty](#) and defaults to *0dp*.

snackbar_y

The snackbar x position in the screen

snackbar_y is a [NumericProperty](#) and defaults to *0dp*.

dismiss(self, *args)

Dismiss the snackbar.

open(self)

Show the snackbar.

on_open(self, *args)

Called when a dialog is opened.

on_dismiss(self, *args)

Called when the dialog is closed.

on_buttons(self, instance, value)

class kivymd.uix.snackbar.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’*.

2.3.42 FitImage

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.uix.fitimage.fitimage

class kivymd.uix.fitimage.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*.

mipmap

Indicate if you want OpenGL mipmapping to be applied to the texture. Read [Mipmapping](#) for more information.

New in version 1.0.0.

mipmap is a [BooleanProperty](#) and defaults to *False*.

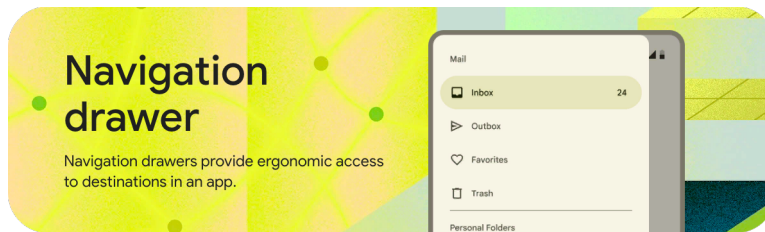
reload(self)

2.3.43 NavigationDrawer

See also:

Material Design 2 spec, Navigation drawer and Material Design 3 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:

Anatomy

```
Root:

    MDNavigationLayout:

        MDScreenManager:

            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 kivymd.uix.boxlayout import MDBoxLayout
from kivymd.app import MDApp

KV = '''
MDScreen:

    MDNavigationLayout:
```

(continues on next page)

(continued from previous page)

```

MDScreenManager:

    MDScreen:

        MDTopAppBar:
            title: "Navigation Drawer"
            elevation: 10
            pos_hint: {"top": 1}
            md_bg_color: "#e7e4c0"
            specific_text_color: "#4a4939"
            left_action_items:
                [['menu', lambda x: nav_drawer.set_state("open")]]

        MDNavigationDrawer:
            id: nav_drawer
            md_bg_color: "#f7f4e7"

        ContentNavigationDrawer:
            ...

class ContentNavigationDrawer(MDBoxLayout):
    pass

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

TestNavigationDrawer().run()

```

Note: *MDNavigationDrawer* is an empty MDCard panel.

Custom content for navigation drawer

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"
    on_release: self.parent.set_color_item(self)

    IconLeftWidget:
        id: icon
        icon: root.icon

```

(continues on next page)

(continued from previous page)

```
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

class DrawerList(ThemableBehavior, MDList):
    def set_color_item(self, instance_item):
```

(continues on next page)

(continued from previous page)

```

'''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])
        )

```

Standard content for the navigation bar

```

from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<DrawerClickableItem@MDNavigationDrawerItem>
    focus_color: "#e7e4c0"
    unfocus_color: "#f7f4e7"
    text_color: "#4a4939"
    icon_color: "#4a4939"
    ripple_color: "#c5bdd2"
    selected_color: "#0c6c4d"

```

(continues on next page)

(continued from previous page)

```

<DrawerLabelItem@MDNavigationDrawerItem>
    bg_color: "#f7f4e7"
    text_color: "#4a4939"
    icon_color: "#4a4939"
    _no_ripple_effect: True

MDScreen:

    MDNavigationLayout:

        MDScreenManager:

            MDScreen:

                MDTopAppBar:
                    title: "Navigation Drawer"
                    elevation: 10
                    pos_hint: {"top": 1}
                    md_bg_color: "#e7e4c0"
                    specific_text_color: "#4a4939"
                    left_action_items:
                        [
                            [
                                nav_drawer.set_state("open")
                                'menu'
                                nav_drawer.set_state("close")
                            ]
                        ]

                MDNavigationDrawer:
                    id: nav_drawer
                    radius: (0, 16, 16, 0) if self.anchor == "left" else (16, 0, 0, 16)
                    md_bg_color: "#f7f4e7"

                    MDNavigationDrawerMenu:

                        MDNavigationDrawerHeader:
                            title: "Header title"
                            title_color: "#4a4939"
                            text: "Header text"
                            spacing: "4dp"
                            padding: "12dp", 0, 0, "56dp"

                        MDNavigationDrawerLabel:
                            text: "Mail"

                        DrawerClickableItem:
                            icon: "gmail"
                            right_text: "+99"
                            text_right_color: "#4a4939"
                            text: "Inbox"

                        DrawerClickableItem:

```

(continues on next page)

(continued from previous page)

```

        icon: "send"
        text: "Outbox"

    MDNavigationDrawerDivider:

    MDNavigationDrawerLabel:
        text: "Labels"

    DrawerLabelItem:
        icon: "information-outline"
        text: "Label"

    DrawerLabelItem:
        icon: "information-outline"
        text: "Label"
'''

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

TestNavigationDrawer().run()

```

Switching screens in the ScreenManager and using the common MDTopAppBar

```

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

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout

KV = '''
<ContentNavigationDrawer>

    ScrollView:

        MDList:

            OneLineListItem:
                text: "Screen 1"
                on_press:
                    root.nav_drawer.set_state("close")
                    root.screen_manager.current = "scr 1"

            OneLineListItem:
                text: "Screen 2"

```

(continues on next page)

(continued from previous page)

```

        on_press:
            root.nav_drawer.set_state("close")
            root.screen_manager.current = "scr 2"

MDScreen:

    MDTopAppBar:
        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

    MDScreenManager:
        id: screen_manager

        MDScreen:
            name: "scr 1"

            MDLabel:
                text: "Screen 1"
                halign: "center"

        MDScreen:
            name: "scr 2"

            MDLabel:
                text: "Screen 2"
                halign: "center"

    MDNavigationDrawer:
        id: nav_drawer

    ContentNavigationDrawer:
        screen_manager: screen_manager
        nav_drawer: nav_drawer
'''

class ContentNavigationDrawer(MDBoxLayout):
    screen_manager = ObjectProperty()
    nav_drawer = ObjectProperty()

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

```

(continues on next page)

(continued from previous page)

```
TestNavigationDrawer().run()
```

API - `kivymd.uix.navigationdrawer.navigationdrawer`

class `kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationLayout(*args, **kwargs)`

Implements the creation and addition of child widgets as declarative programming style.

update_pos(*self*, *instance_navigation_drawer*, *pos_x*: *float*)

add_scrim(*self*, *instance_manager*: *ScreenManager*)

update_scrim_rectangle(*self*, *instance_manager*: *ScreenManager*, *size*: *list*)

add_widget(*self*, *widget*, *index*=0, *canvas*=None)

Only two layouts are allowed: `ScreenManager` and `MDNavigationDrawer`.

class `kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerLabel(*args, **kwargs)`

Implements a label for a menu for `MDNavigationDrawer` class.

New in version 1.0.0.

```
MDNavigationDrawer:
```

```
    MDNavigationDrawerMenu:
```

```
        MDNavigationDrawerLabel:
```

```
            text: "Mail"
```


**text**

Text label.

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

padding

Padding between layout box and children: [padding_left, padding_top, padding_right, padding_bottom].

Padding also accepts a two argument form [padding_horizontal, padding_vertical] and a one argument form [padding].

`padding` is a `VariableListProperty` and defaults to `['20dp', 0, 0, '8dp']`.

```
class kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerDivider(*args,
                                                                              **kwargs)
```

Implements a divider for a menu for `MDNavigationDrawer` class.

New in version 1.0.0.

```
MDNavigationDrawer:
    MDNavigationDrawerMenu:
        MDNavigationDrawerLabel:
            text: "Mail"
        MDNavigationDrawerDivider:
```

**padding**

Padding between layout box and children: [padding_left, padding_top, padding_right, padding_bottom].

Padding also accepts a two argument form [padding_horizontal, padding_vertical] and a one argument form [padding].

padding is a [VariableListProperty](#) and defaults to ['20dp', '12dp', 0, '12dp'].

color

Divider color in rgba format.

color is a [ColorProperty](#) and defaults to *None*.

class kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader(**kwargs)

Implements a header for a menu for [MDNavigationDrawer](#) class.

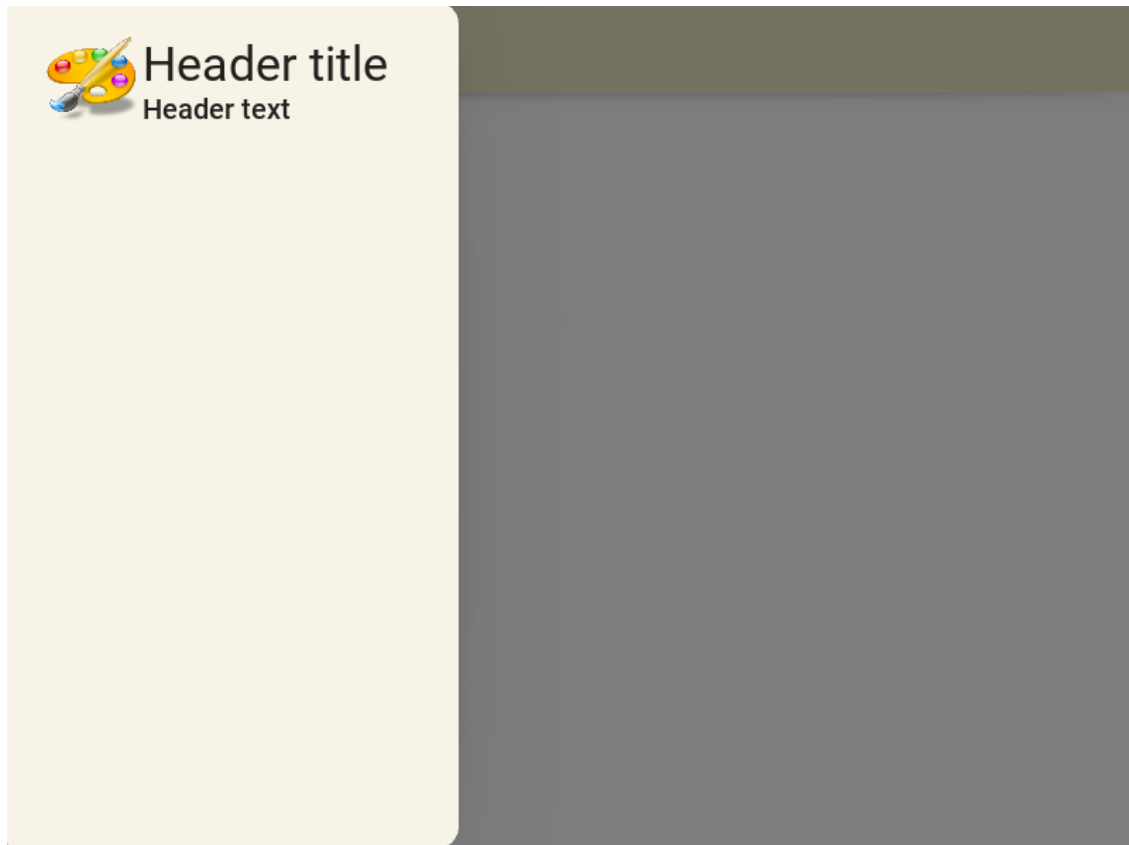
New in version 1.0.0.

```
MDNavigationDrawer:
    MDNavigationDrawerMenu:
        MDNavigationDrawerHeader:
            title: "Header title"
            text: "Header text"
            spacing: "4dp"
            padding: "12dp", 0, 0, "56dp"
```

**source**

Image logo path.

```
MDNavigationDrawer:
    MDNavigationDrawerMenu:
        MDNavigationDrawerHeader:
            title: "Header title"
            text: "Header text"
            source: "logo.png"
            spacing: "4dp"
            padding: "12dp", 0, 0, "56dp"
```



`source` is a `StringProperty` and defaults to `''`.

title

Title shown in the first line.

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

title_halign

Title halign first line.

`title_halign` is a `StringProperty` and defaults to `'left'`.

title_color

Title text color.

`title_color` is a `ColorProperty` and defaults to `None`.

title_font_style

Title shown in the first line.

`title_font_style` is a `StringProperty` and defaults to `'H4'`.

title_font_size

Title shown in the first line.

`title_font_size` is a `StringProperty` and defaults to `'34sp'`.

text

Text shown in the second line.

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

text_halign

Text halign first line.

`text_halign` is a `StringProperty` and defaults to `'left'`.

text_color

Title text color.

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

text_font_style

Title shown in the first line.

`text_font_style` is a `StringProperty` and defaults to `'H6'`.

text_font_size

Title shown in the first line.

`text_font_size` is a `StringProperty` and defaults to `'20sp'`.

check_content(*self*, *interval*: *Union[int, float]*)

Removes widgets that the user has not added to the container.

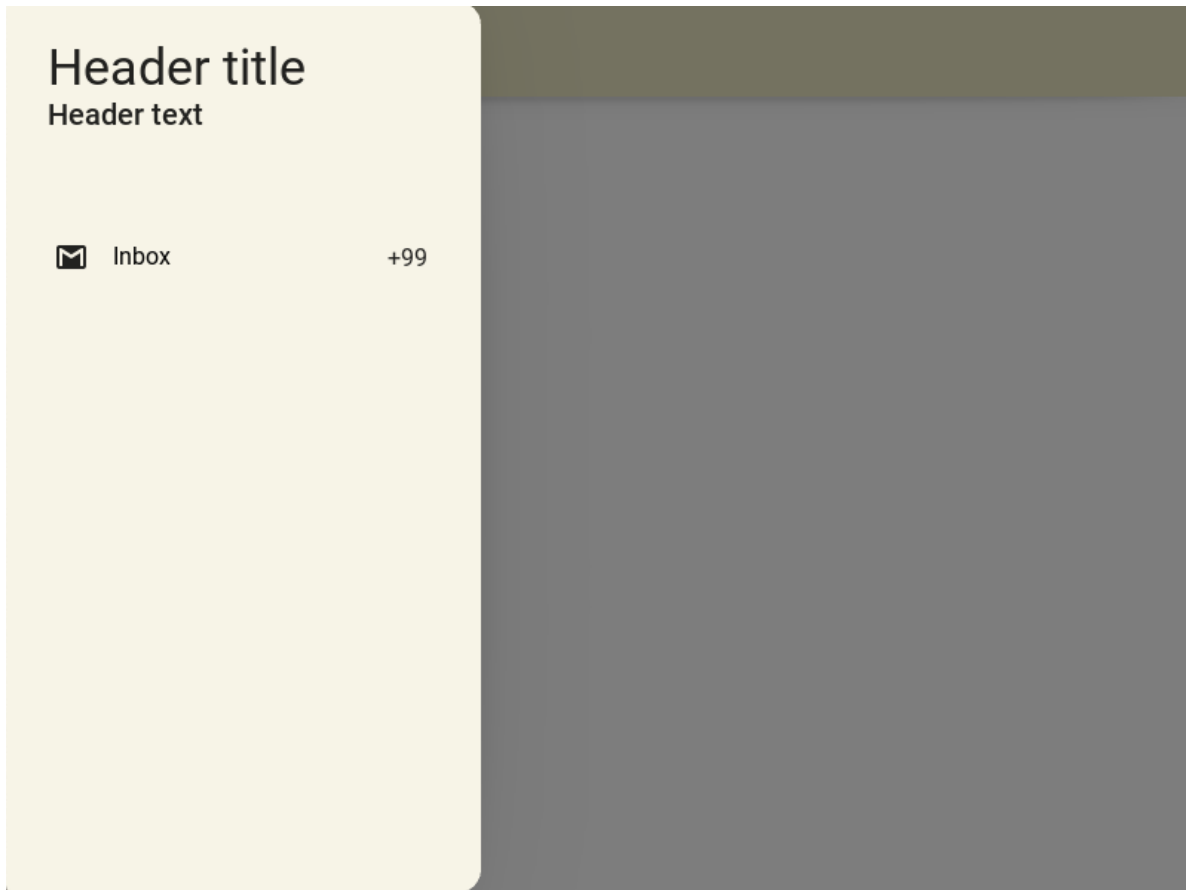
class `kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerItem(*args, **kwargs)`

Implements an item for the `MDNavigationDrawer` menu list.

New in version 1.0.0.

```
MDNavigationDrawer:
    MDNavigationDrawerMenu:
        MDNavigationDrawerHeader:
            title: "Header title"
            text: "Header text"
            spacing: "4dp"
            padding: "12dp", 0, 0, "56dp"

        MDNavigationDrawerItem
            icon: "gmail"
            right_text: "+99"
            text: "Inbox"
```

**selected**

Is the item selected.

selected is a `BooleanProperty` and defaults to *False*.

icon

Icon item.

icon is a `StringProperty` and defaults to *''*.

icon_color

Icon color item.

icon_color is a `ColorProperty` and defaults to *None*.

selected_color

The color of the icon and text of the selected item.

selected_color is a `ColorProperty` and defaults to *[0, 0, 0, 1]*.

right_text

Right text item.

right_text is a `StringProperty` and defaults to *''*.

text_right_color

Right text color item.

text_right_color is a `ColorProperty` and defaults to *None*.

class kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerMenu(*args, **kwargs)

Implements a scrollable list for menu items of the [MDNavigationDrawer](#) class.

New in version 1.0.0.

```
MDNavigationDrawer:

    MDNavigationDrawerMenu:

        # Your menu items.
        ...
```

spacing

Spacing between children, in pixels.

[spacing](#) is a [NumericProperty](#) and defaults to 0.

add_widget(self, widget, *args, **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)
```

reset_active_color(self, item: [MDNavigationDrawerItem](#))

class kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer(*args, **kwargs)

Implements the creation and addition of child widgets as declarative programming style.

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.

```
MDNavigationDrawer:
    type: "standard"
```

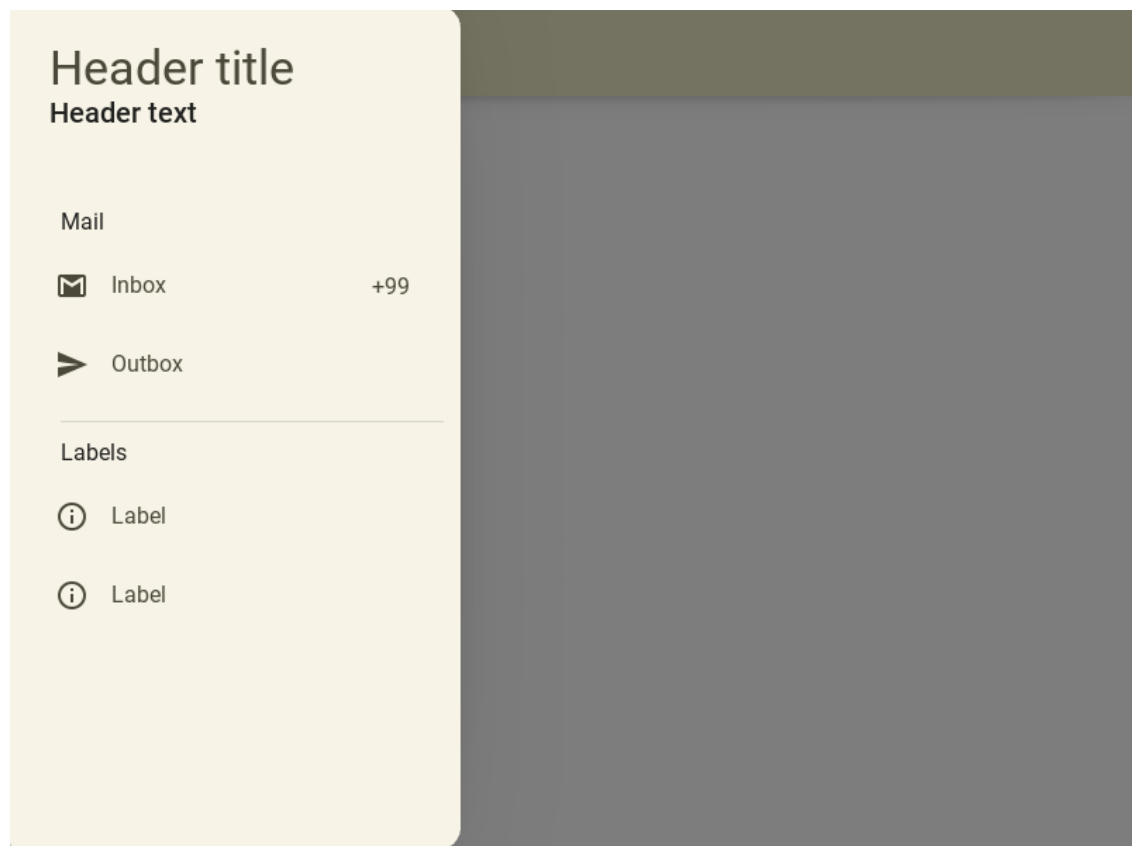
```
MDNavigationDrawer:  
    type: "modal"
```

`type` is a `OptionProperty` and defaults to `'modal'`.

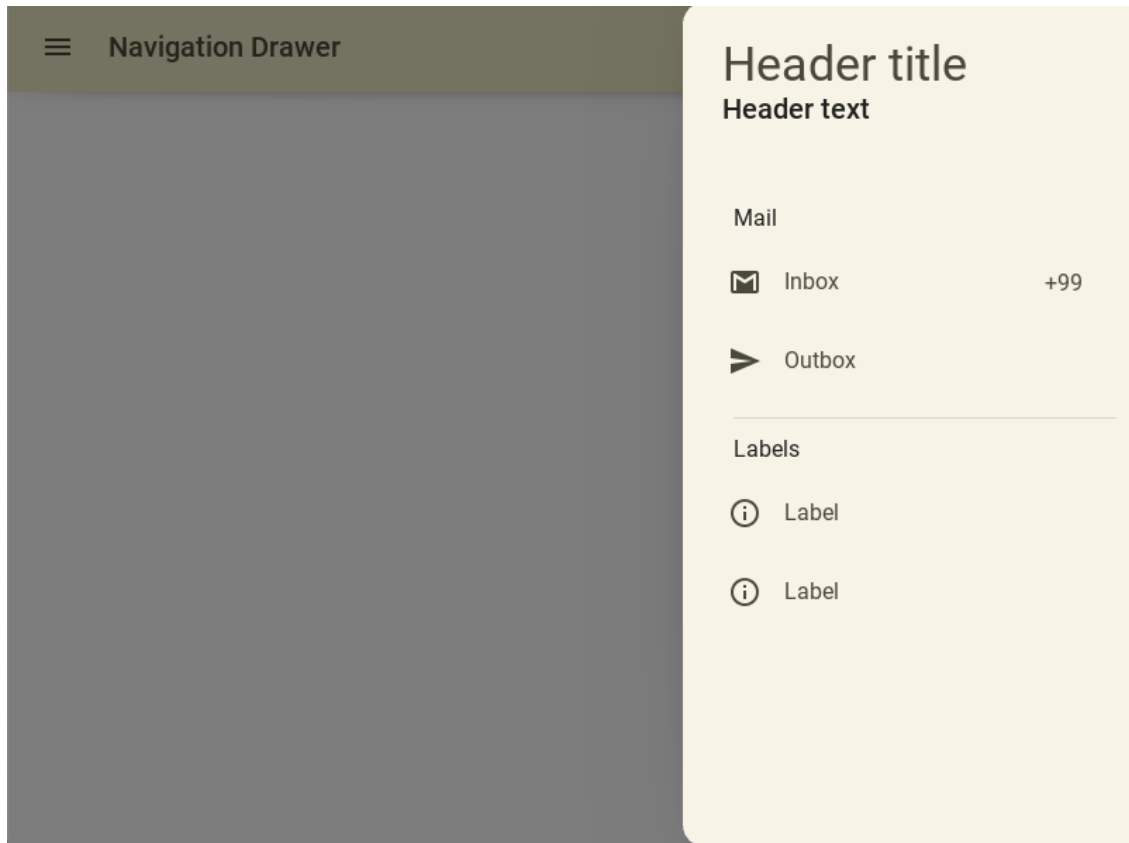
anchor

Anchoring screen edge for drawer. Set it to `'right'` for right-to-left languages. Available options are: `'left'`, `'right'`.

```
MDNavigationDrawer:  
    anchor: "left"
```



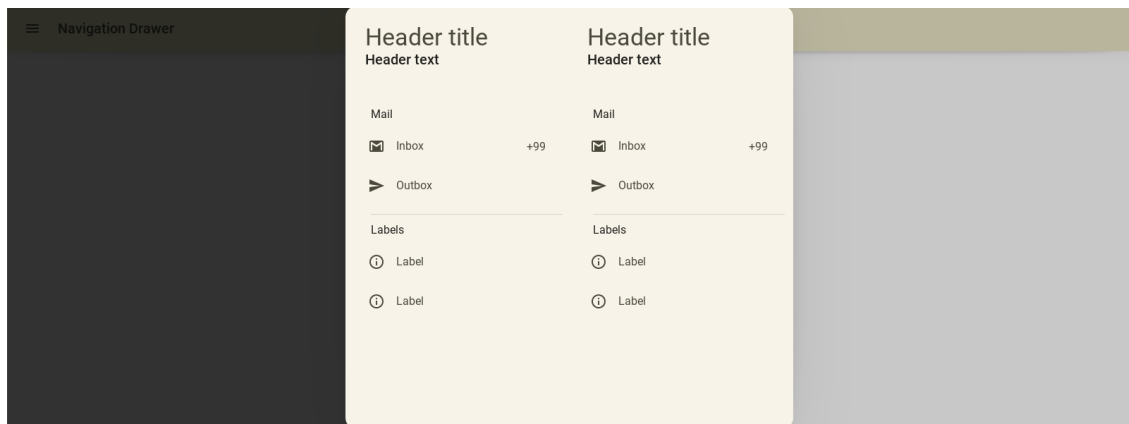
```
MDNavigationDrawer:  
    anchor: "right"
```

`anchor` is a `OptionProperty` and defaults to `'left'`.

scrim_color

Color for scrim. Alpha channel will be multiplied with `_scrim_alpha`. Set fourth channel to 0 if you want to disable scrim.



```
MDNavigationDrawer:
    scrim_color: 0, 0, 0, .8
    # scrim_color: 0, 0, 0, .2
```

`scrim_color` is a `ColorProperty` and defaults to `[0, 0, 0, 0.5]`.

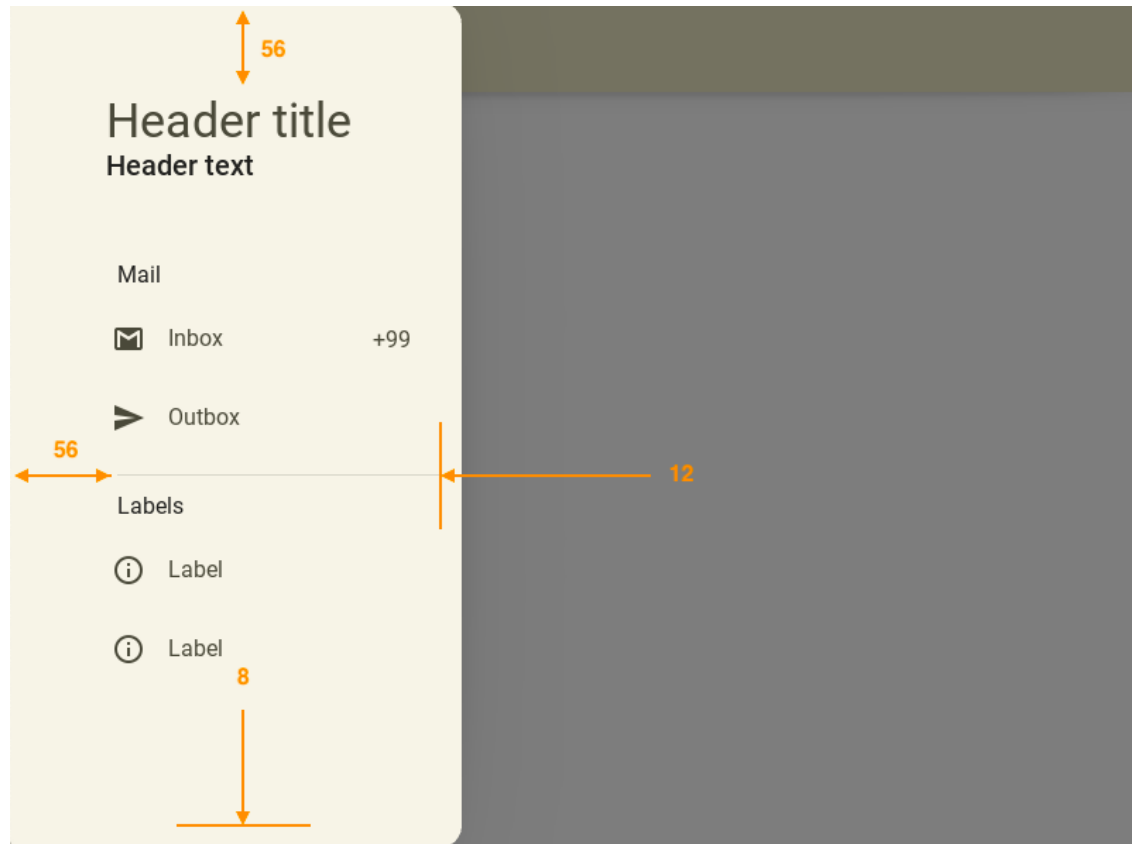
padding

Padding between layout box and children: `[padding_left, padding_top, padding_right, padding_bottom]`.

Padding also accepts a two argument form [padding_horizontal, padding_vertical] and a one argument form [padding].

Changed in version 1.0.0.

```
MDNavigationDrawer:
    padding: 56, 56, 12, 16
```



`padding` is a `VariableListProperty` and defaults to `'[16, 16, 12, 16]'`.

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_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: float)

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_navigation_drawer*, *radius_value*: *list*)

on_type(*self*, *instance_navigation_drawer*, *drawer_type*: *str*)

2.3.44 TextField

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}
```

(continues on next page)

(continued from previous page)

```
'''

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,
        )
        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"
```

Rectangle mode

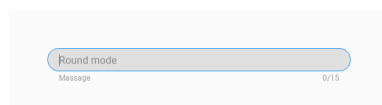
```
MDTextField:
    hint_text: "Rectangle mode"
    mode: "rectangle"
```

Fill mode

```
MDTextField:
    hint_text: "Fill mode"
    mode: "fill"
```

Round mode

```
MDTextField:
    hint_text: "Round mode"
    mode: "round"
    max_text_length: 15
    helper_text: "Message"
```



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.

Clickable icon for MDTextField

```
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

    MDTextField:
        id: text_field
        hint_text: root.hint_text
        text: root.text
        password: True
        icon_left: "key-variant"

    MDIconButton:
        icon: "eye-off"
        pos_hint: {"center_y": .5}
        pos: text_field.width - self.width + dp(8), 0
        theme_text_color: "Hint"
        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:
    # [...]

class Test(MDApp):
```

(continues on next page)

(continued from previous page)

```
def build(self):
    return Builder.load_string(KV)
```

```
Test().run()
```

See also:

See more information in the [MDTextFieldRect](#) class.

API - kivymd.uix.textfield.textfield

class kivymd.uix.textfield.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.

Changed in version 2.1.0: `keyboard_suggestions` is now inherited from `FocusBehavior`.

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.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.

Changed in version 2.1.0: `keyboard_suggestions` is now inherited from `FocusBehavior`.

helper_text

Text for `helper_text` mode.

`helper_text` is an `StringProperty` and defaults to ‘’.

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`.

mode

Text field mode. Available options are: `'line'`, `'rectangle'`, `'fill'`, `'round'`.

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

line_color_normal

Line color normal (static underline line) in rgba format.

```
MDTextField:
    hint_text: "line_color_normal"
    line_color_normal: 1, 0, 1, 1
```

`line_color_normal` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

line_color_focus

Line color focus (active underline line) in rgba format.

```
MDTextField:
    hint_text: "line_color_focus"
    line_color_focus: 0, 1, 0, 1
```

`line_color_focus` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

line_anim

If True, then text field shows animated underline 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 `[0, 0, 0, 0]`.

fill_color_normal

Fill background color in 'fill' mode when text field is out of focus.

`fill_color_normal` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

fill_color_focus

Fill background color in 'fill' mode when the text field has focus.

`fill_color_focus` 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`.

hint_text_color_normal

Hint text color when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    hint_text: "hint_text_color_normal"
    hint_text_color_normal: 0, 1, 0, 1
```

`hint_text_color_normal` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

hint_text_color_focus

Hint text color when the text field has focus.

New in version 1.0.0.

```
MDTextField:
    hint_text: "hint_text_color_focus"
    hint_text_color_focus: 0, 1, 0, 1
```

`hint_text_color_focus` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

helper_text_color_normal

Helper text color when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    helper_text: "helper_text_color_normal"
    helper_text_mode: "persistent"
    helper_text_color_normal: 0, 1, 0, 1
```



`helper_text_color_normal` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

helper_text_color_focus

Helper text color when the text field has focus.

New in version 1.0.0.

```
MDTextField:
    helper_text: "helper_text_color_focus"
    helper_text_mode: "persistent"
    helper_text_color_focus: 0, 1, 0, 1
```

helper_text_color_focus is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

icon_right_color_normal

Color of right icon when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    icon_right: "language-python"
    hint_text: "icon_right_color_normal"
    icon_right_color_normal: 0, 1, 0, 1
```

icon_right_color_normal is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

icon_right_color_focus

Color of right icon when the text field has focus.

New in version 1.0.0.

```
MDTextField:
    icon_right: "language-python"
    hint_text: "icon_right_color_focus"
    icon_right_color_focus: 0, 1, 0, 1
```

icon_right_color_focus is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

icon_left_color_normal

Color of right icon when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    icon_right: "language-python"
    hint_text: "icon_right_color_normal"
    icon_left_color_normal: 0, 1, 0, 1
```

icon_left_color_normal is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

icon_left_color_focus

Color of right icon when the text field has focus.

New in version 1.0.0.

```
MDTextField:
    icon_right: "language-python"
    hint_text: "icon_right_color_focus"
    icon_right_color_focus: 0, 1, 0, 1
```

icon_left_color_focus is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

max_length_text_color

Text color of the maximum length of characters to be input.

New in version 1.0.0.

```
MDTextField:
    hint_text: "max_length_text_color"
    max_length_text_color: 0, 1, 0, 1
    max_text_length: 5
```

max_length_text_color is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

icon_right

Right icon texture.

Note: It's just a texture. It has no press/touch events.

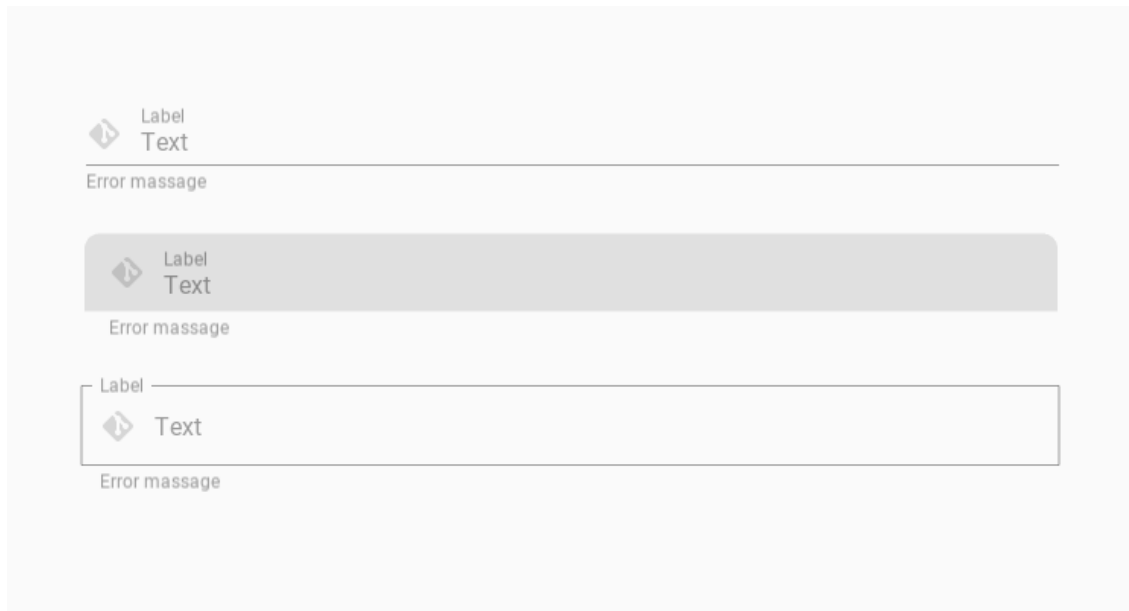
icon_right is an `StringProperty` and defaults to `''`.

icon_left

Left icon texture.

New in version 1.0.0.

Note: It's just a texture. It has no press/touch events. Also note that you cannot use the left and right icons at the same time yet.



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

text_color_normal

Text color in `rgba` format when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    hint_text: "text_color_normal"
    text_color_normal: 0, 1, 0, 1
```

`text_color_normal` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

text_color_focus

Text color in `rgba` format when text field has focus.

New in version 1.0.0.

```
MDTextField:
    hint_text: "text_color_focus"
    text_color_focus: 0, 1, 0, 1
```

`text_color_focus` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

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`.

```
MDTextField:
    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}
```

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'`.

cancel_all_animations_on_double_click(self)

Cancels the animations of the text field when double-clicking on the text field.

set_colors_to_updated(self, interval: Union[float, int])

set_default_colors(self, interval: Union[float, int], updated: bool = False)

Sets the default text field colors when initializing a text field object. Also called when the application palette changes.

Parameters

updated – If `True` - the color theme of the application has been changed. Updating the meanings of the colors.

set_notch_rectangle(self, joining: bool = False)

Animates a notch for the hint text in the rectangle of the text field of type *rectangle*.

set_active_underline_width(self, width: Union[float, int])

Animates the width of the active underline line.

set_static_underline_color(self, color: list)

Animates the color of a static underline line.

set_active_underline_color(self, color: list)

Animates the fill color for `'fill'` mode.

set_fill_color(*self*, *color*: *list*)
 Animates the color of the hint text.

set_helper_text_color(*self*, *color*: *list*)
 Animates the color of the hint text.

set_max_length_text_color(*self*, *color*: *list*)
 Animates the color of the max length text.

set_icon_right_color(*self*, *color*: *list*)
 Animates the color of the icon right.

set_icon_left_color(*self*, *color*: *list*)
 Animates the color of the icon left.

set_hint_text_color(*self*, *focus*: *bool*, *error*: *bool* = *False*)
 Animates the color of the hint text.

set_pos_hint_text(*self*, *y*: *float*, *x*: *float* = 12)
 Animates the x-axis width and y-axis height of the hint text.

set_hint_text_font_size(*self*, *font_size*: *float*)
 Animates the font size of the hint text.

set_max_text_length(*self*)
 Called when text is entered into a text field.

check_text(*self*, *interval*: *Union[float, int]*)

set_text(*self*, *instance_text_field*, *text*: *str*)
 Called when text is entered into a text field.

set_x_pos(*self*)

set_objects_labels(*self*)
 Creates labels objects for the parameters`helper_text`, `hint_text`, etc.

on_helper_text(*self*, *instance_text_field*, *helper_text*: *str*)

on_focus(*self*, *instance_text_field*, *focus*: *bool*)

on_icon_left(*self*, *instance_text_field*, *icon_name*: *str*)

on_icon_right(*self*, *instance_text_field*, *icon_name*: *str*)

on_disabled(*self*, *instance_text_field*, *disabled_value*: *bool*)

on_error(*self*, *instance_text_field*, *error*: *bool*)
 Changes the primary colors of the text box to match the *error* value (text field is in an error state or not).

on_hint_text(*self*, *instance_text_field*, *hint_text*: *str*)

on_width(*self*, *instance_text_field*, *width*: *float*)
 Called when the application window is resized.

on_height(*self*, *instance_text_field*, *value_height*: *float*)

on_text_color_normal(*self*, *instance_text_field*, *color*: *list*)

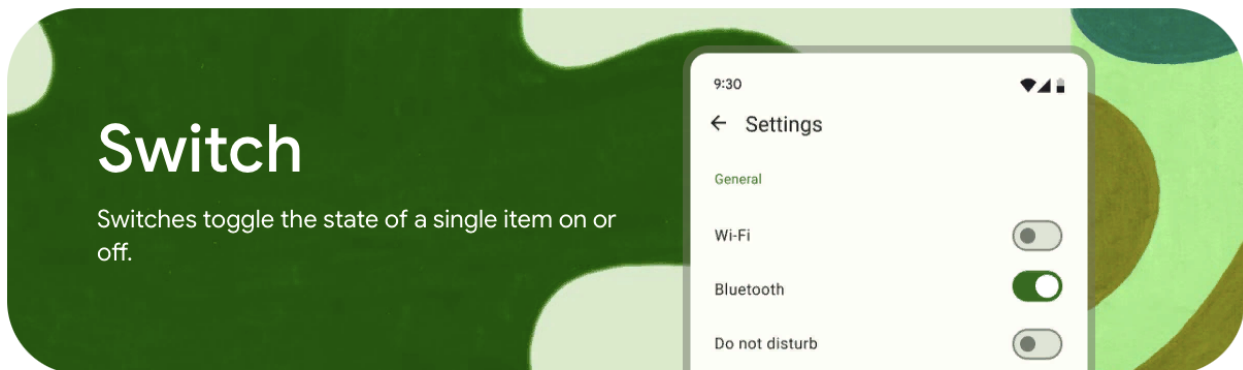
```
on_hint_text_color_normal(self, instance_text_field, color: list)
on_helper_text_color_normal(self, instance_text_field, color: list)
on_icon_right_color_normal(self, instance_text_field, color: list)
on_line_color_normal(self, instance_text_field, color: list)
on_max_length_text_color(self, instance_text_field, color: list)
```

2.3.45 SelectionControls

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}
```

(continues on next page)

(continued from previous page)

```
'''

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}
```

(continues on next page)

(continued from previous page)

```
'''

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:

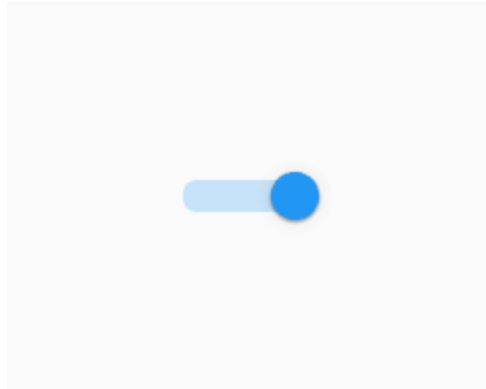
    MDSwitch:
        pos_hint: {'center_x': .5, 'center_y': .5}
'''

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*.

MDSwitch in M3 style

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDSwitch:
        pos_hint: {'center_x': .5, 'center_y': .5}
        active: True
'''

class Test(MDApp):
    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(KV)

Test().run()
```

API - `kivymd.ui.selectioncontrol.selectioncontrol`

class `kivymd.ui.selectioncontrol.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.

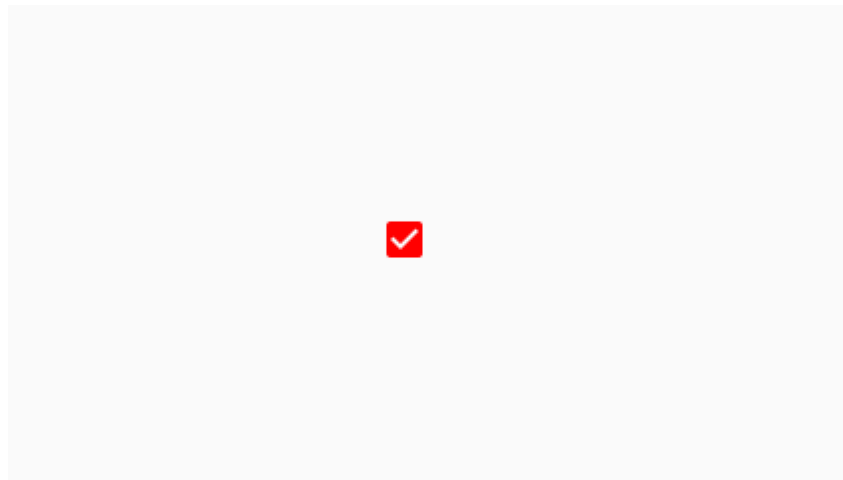
`radio_icon_down` is a `StringProperty` and defaults to `'checkbox-marked-circle'`.

color_active

Color when the checkbox is in the active state.

New in version 1.0.0.

```
MDCheckbox:
    color_active: "red"
```



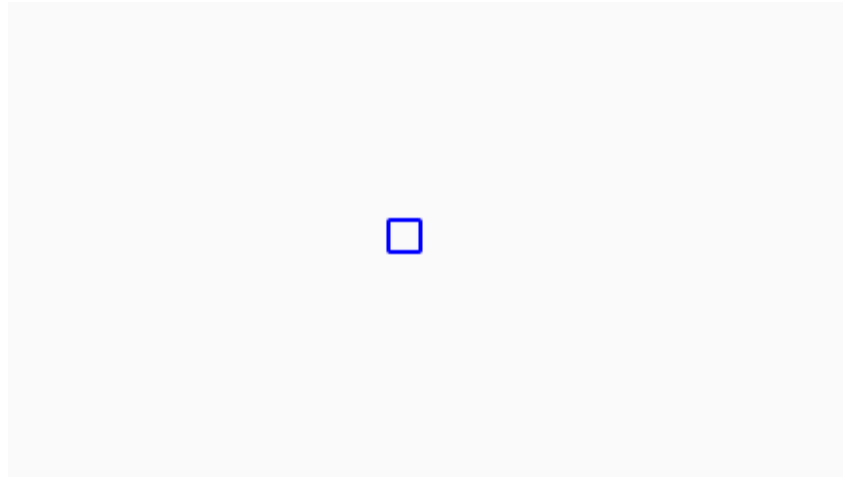
`color_active` is a `ColorProperty` and defaults to `None`.

color_inactive

Color when the checkbox is in the inactive state.

New in version 1.0.0.

```
MDCheckbox:
    color_inactive: "blue"
```

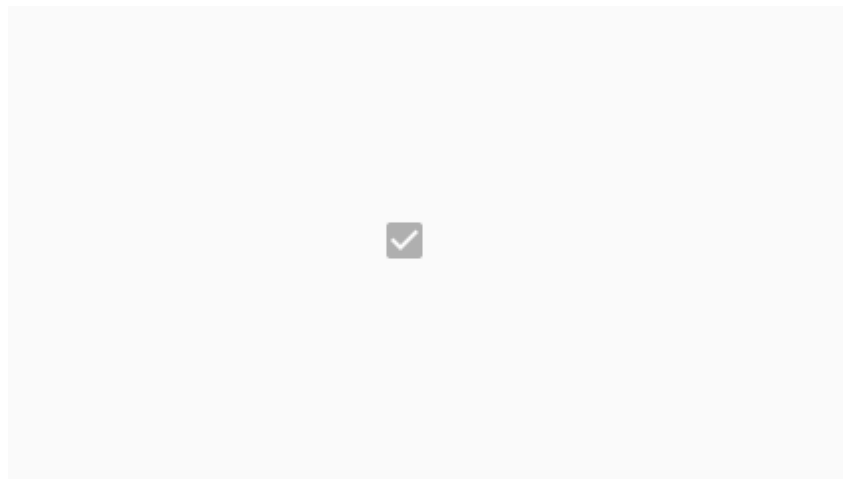


`color_inactive` is a `ColorProperty` and defaults to `None`.

disabled_color

Color when the checkbox is in the disabled state.

```
MDCheckbox:
    disabled_color: "lightgrey"
    disabled: True
    active: True
```



`disabled_color` is a `ColorProperty` and defaults to `None`.

selected_color

Color when the checkbox is in the active state.

Deprecated since version 1.0.0: Use `color_active` instead.

`selected_color` is a `ColorProperty` and defaults to `None`.

unselected_color

Color when the checkbox is in the inactive state.

Deprecated since version 1.0.0: Use `color_inactive` instead.

`unselected_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.selectioncontrol.MDSwitch(**kwargs)

Float layout class. See module documentation for more information.

active

Indicates if the switch is active or inactive.

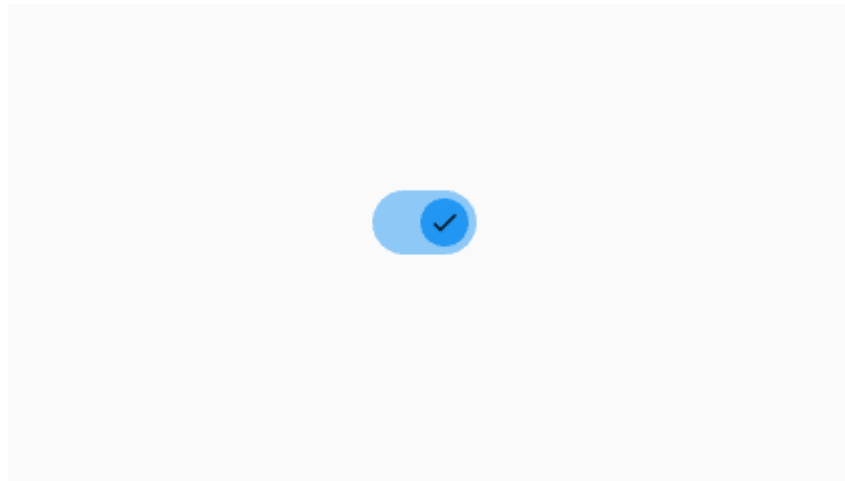
active is a `BooleanProperty` and defaults to *False*.

icon_active

Thumb icon when the switch is in the active state (only M3 style).

New in version 1.0.0.

```
MDSwitch:
    active: True
    icon_active: "check"
```



icon_active is a `StringProperty` and defaults to `''`.

icon_inactive

Thumb icon when the switch is in an inactive state (only M3 style).

New in version 1.0.0.

```
MDSwitch:
    icon_inactive: "close"
```



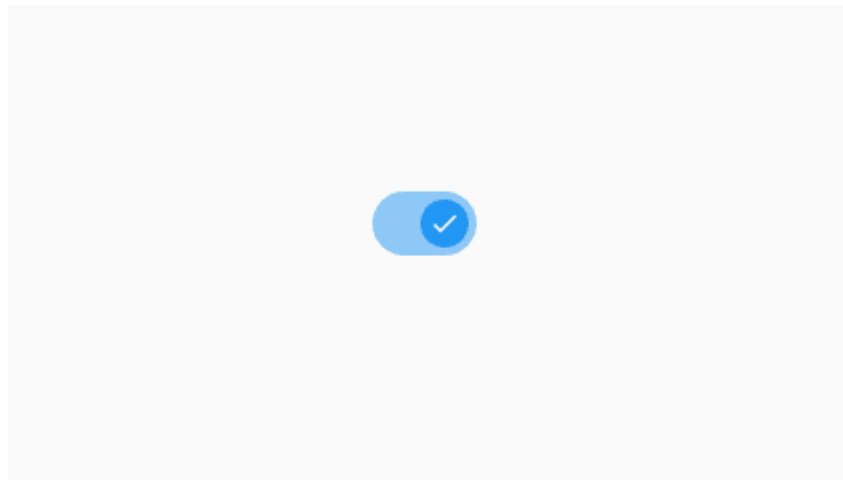

`icon_inactive` is a `StringProperty` and defaults to `'X'`.

icon_active_color

Thumb icon color when the switch is in the active state (only M3 style).

New in version 1.0.0.

```
MDSwitch:
    active: True
    icon_active: "check"
    icon_active_color: "white"
```



`icon_active_color` is a `ColorProperty` and defaults to `None`.

icon_inactive_color

Thumb icon color when the switch is in an inactive state (only M3 style).

New in version 1.0.0.

```
MDSwitch:
    icon_inactive: "close"
    icon_inactive_color: "grey"
```



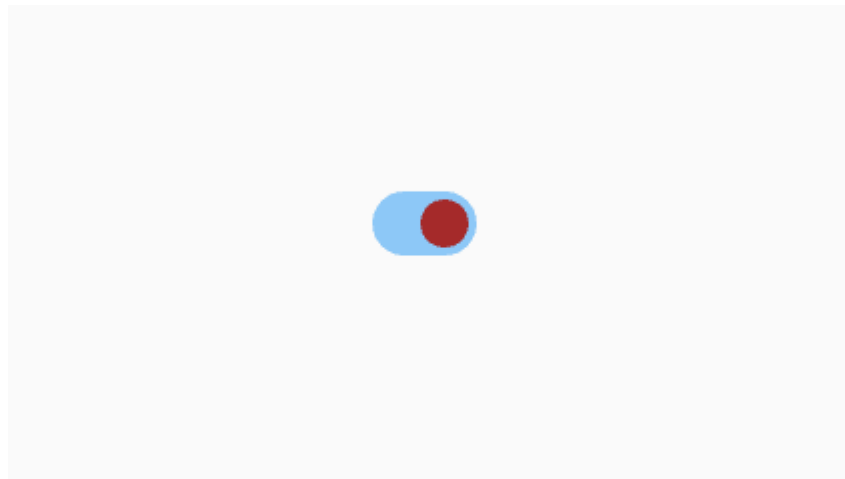
`icon_inactive_color` is a `ColorProperty` and defaults to `None`.

thumb_color_active

The color of the thumb when the switch is active.

New in version 1.0.0.

```
MDSwitch:
    active: True
    thumb_color_active: "brown"
```



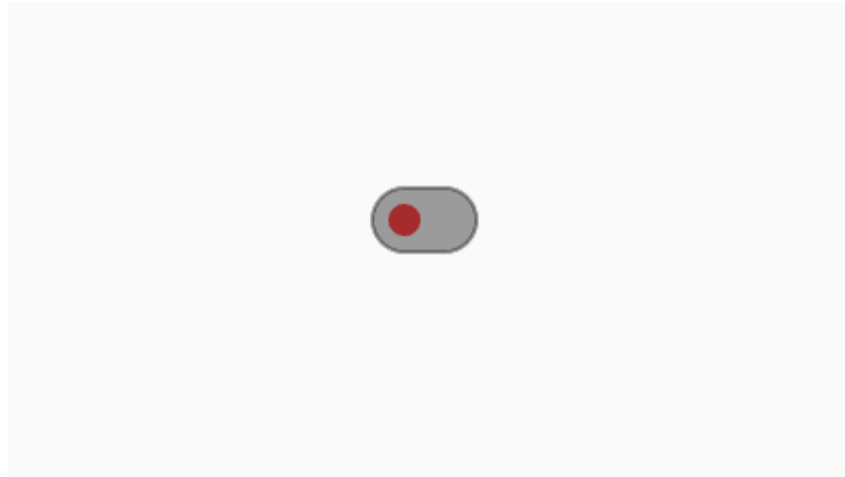
`thumb_color_active` is an `ColorProperty` and default to `None`.

thumb_color_inactive

The color of the thumb when the switch is inactive.

New in version 1.0.0.

```
MDSwitch:
    thumb_color_inactive: "brown"
```

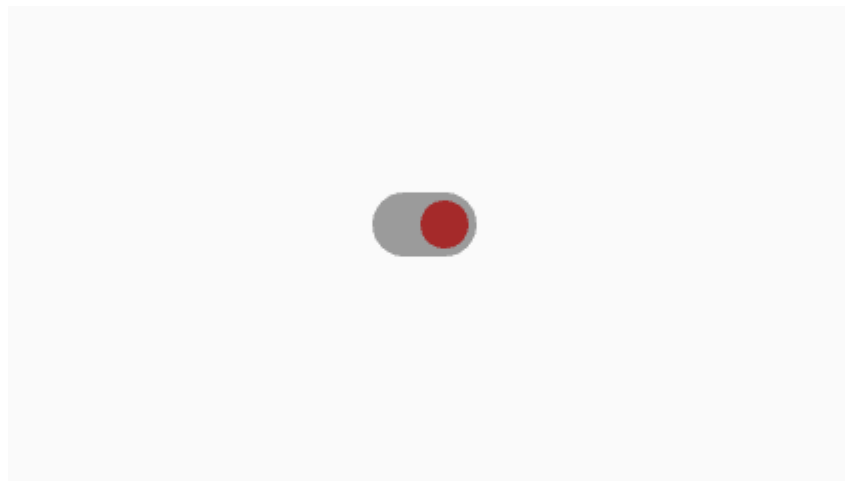


`thumb_color_inactive` is an `ColorProperty` and default to `None`.

thumb_color_disabled

The color of the thumb when the switch is in the disabled state.

```
MDSwitch:
    active: True
    thumb_color_disabled: "brown"
    disabled: True
```

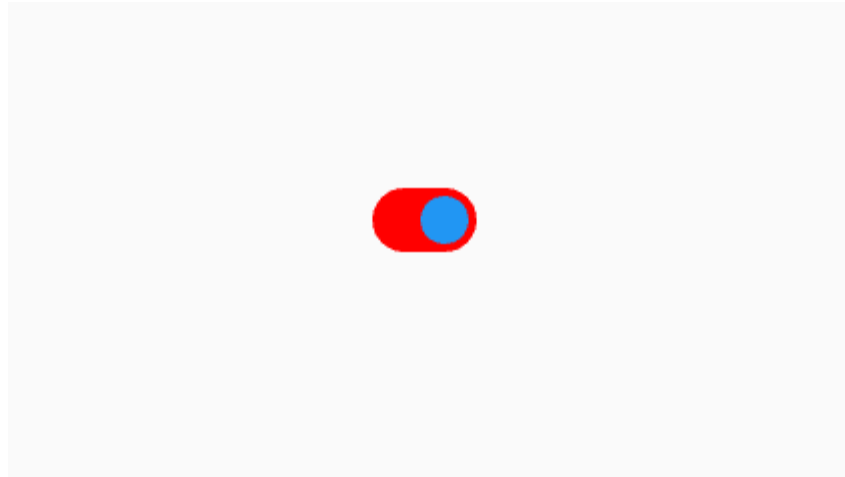


`thumb_color_disabled` is an `ColorProperty` and default to `None`.

track_color_active

The color of the track when the switch is active.

```
MDSwitch:
    active: True
    track_color_active: "red"
```



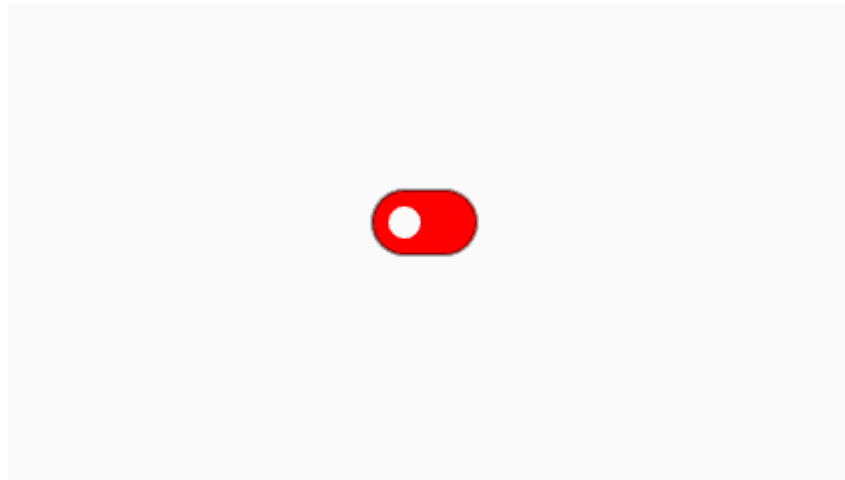
`track_color_active` is an `ColorProperty` and default to *None*.

track_color_inactive

The color of the track when the switch is inactive.

New in version 1.0.0.

```
MDSwitch:  
    track_color_inactive: "red"
```



`track_color_inactive` is an `ColorProperty` and default to *None*.

track_color_disabled

The color of the track when the switch is in the disabled state.

```
MDSwitch:  
    track_color_disabled: "lightgrey"  
    disabled: True
```



`track_color_disabled` is an `ColorProperty` and default to `None`.

set_icon(*self*, *instance_switch*, *icon_value*: *str*)

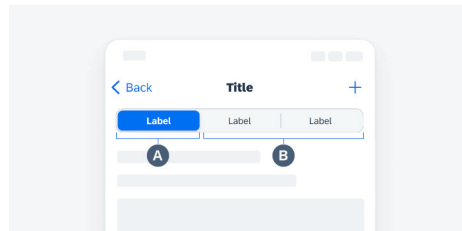
on_active(*self*, *instance_switch*, *active_value*: *bool*)

on_thumb_down(*self*)

Called at the `on_touch_down` event of the `Thumb` object. Indicates the state of the switch “on/off” by an animation of increasing the size of the thumb.

2.3.46 SegmentedControl

New in version 1.0.0.



Usage

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

KV = '''
MDScreen:
    MDSegmentedControl:
        pos_hint: {"center_x": .5, "center_y": .5}
```

(continues on next page)

(continued from previous page)

```

        MDSegmentedControlItem:
            text: "Male"

        MDSegmentedControlItem:
            text: "Female"

        MDSegmentedControlItem:
            text: "All"
    ...

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

Test().run()

```

Or only in python code:

```

from kivymd.app import MDApp
from kivymd.ui.screen import MDScreen
from kivymd.ui.segmentedcontrol import MDSegmentedControl, MDSegmentedControlItem

class Test(MDApp):
    def build(self):
        screen = MDScreen()
        segment_control = MDSegmentedControl(pos_hint={"center_x": .5, "center_y": .5})
        segment_control.add_widget(MDSegmentedControlItem(text="Male"))
        segment_control.add_widget(MDSegmentedControlItem(text="Female"))
        segment_control.add_widget(MDSegmentedControlItem(text="All"))
        screen.add_widget(segment_control)
        return screen

Test().run()

```

Events

```

MDSegmentedControl:
    on_active: app.on_active(*args)

```

```

def on_active(
    self,
    segmented_control: MDSegmentedControl,
    segmented_item: MDSegmentedControlItem,
) -> None:
    '''Called when the segment is activated.'''

```

API - `kivymd.uix.segmentedcontrol.segmentedcontrol`

class `kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControlItem(**kwargs)`

Implements a label to place on the SegmentPanel panel.

class `kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl(*args, **kwargs)`

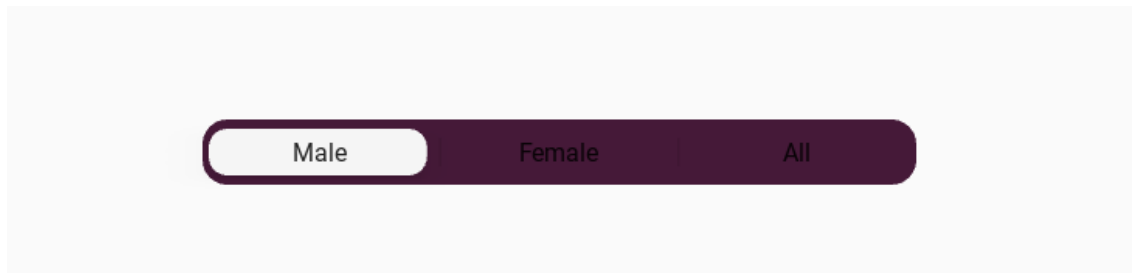
Events***on_active***

Called when the segment is activated.

`md_bg_color`

Background color of the segment panel.

```
MDSegmentedControl:
    md_bg_color: "#451938"
```

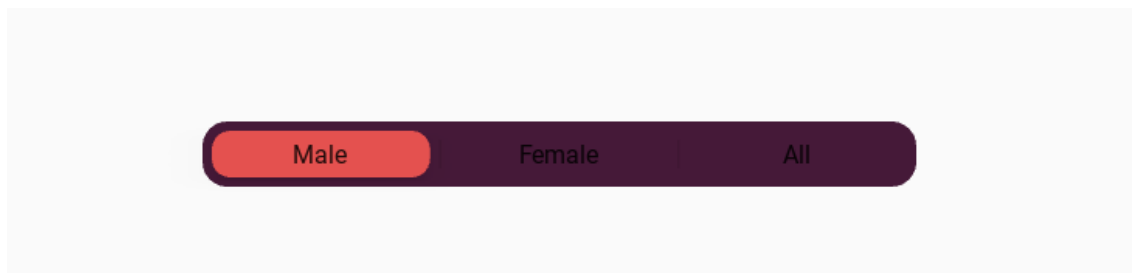


`md_bg_color` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

`segment_color`

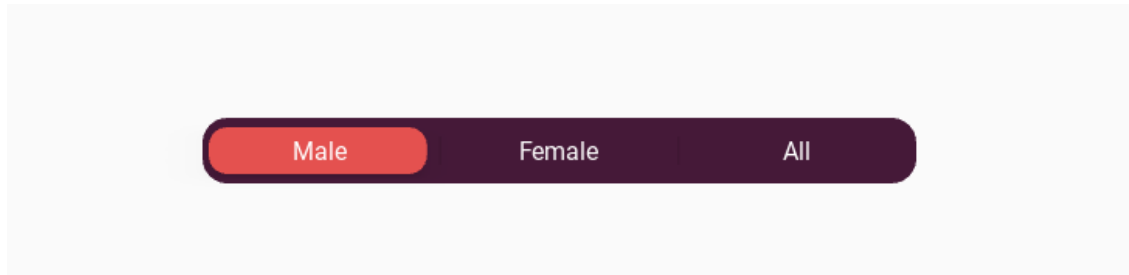
Color of the active segment.

```
MDSegmentedControl:
    md_bg_color: "#451938"
    segment_color: "#e4514f"
```



```
MDSegmentedControl:
    md_bg_color: "#451938"
    segment_color: "#e4514f"

    MDSegmentedControlItem:
        text: "[color=fff]Male[/color]"
```

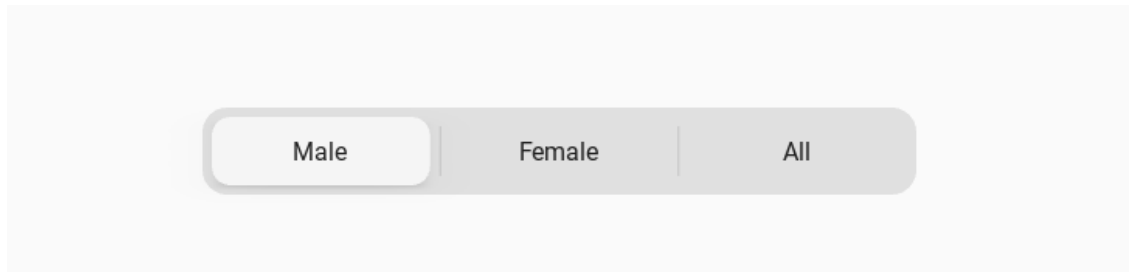


`segment_color` is an `ColorProperty` and defaults to `[0, 0, 0, 0]`.

segment_panel_height

Height of the segment panel.

```
MDSegmentedControl:
    segment_panel_height: "56dp"
```

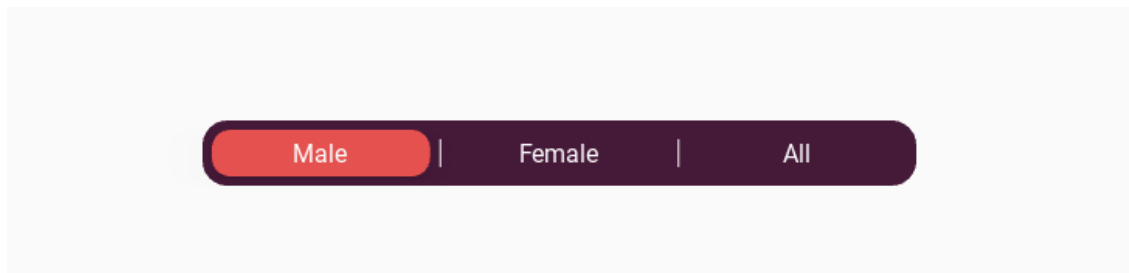


`segment_panel_height` is an `NumericProperty` and defaults to `'42dp'`.

separator_color

The color of the separator between the segments.

```
MDSegmentedControl:
    md_bg_color: "#451938"
    segment_color: "#e4514f"
    separator_color: 1, 1, 1, 1
```

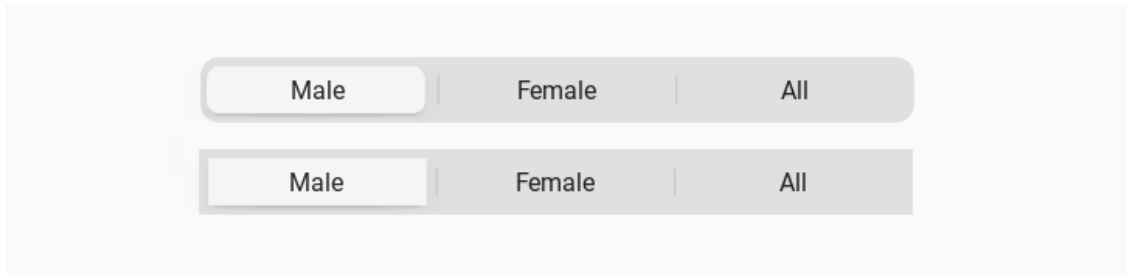


`separator_color` is an `ColorProperty` and defaults to `None`.

radius

Radius of the segment panel.

```
MDSegmentedControl:
    radius: 0
```

radius is an `VariableListProperty` and defaults to `[16, 16, 16, 16]`.

segment_switching_transition

Name of the animation type for the switch segment.

segment_switching_transition is a `StringProperty` and defaults to `'in_cubic'`.

segment_switching_duration

Name of the animation type for the switch segment.

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

current_active_segment

The current active element of the `MDSegmentedControlItem` class.

current_active_segment is a `ObjectProperty` and defaults to `None`.

set_default_colors(self, *args)

Sets the colors of the panel and the switch if the colors are not set by the user.

animation_segment_switch(self, widget: MDSegmentedControlItem)

Animates the movement of the switch.

update_segment_panel_width(self, widget: MDSegmentedControlItem)

Sets the width of the panel for the elements of the `MDSegmentedControlItem` class.

update_separator_color(self, widget: MDSeparator)

Updates the color of the separators between segments.

add_widget(self, widget, *args, **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)
```

on_active(*self*, *args)

Called when the segment is activated.

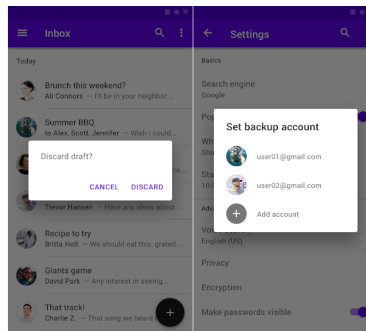
on_press_segment(*self*, widget: MDSegmentedControlItem, touch)

2.3.47 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 MDFlatButton
from kivymd.uix.dialog import MDDialog

KV = '''
MDFloatLayout:

    MDFlatButton:
        text: "ALERT DIALOG"
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_alert_dialog()
'''
```

(continues on next page)

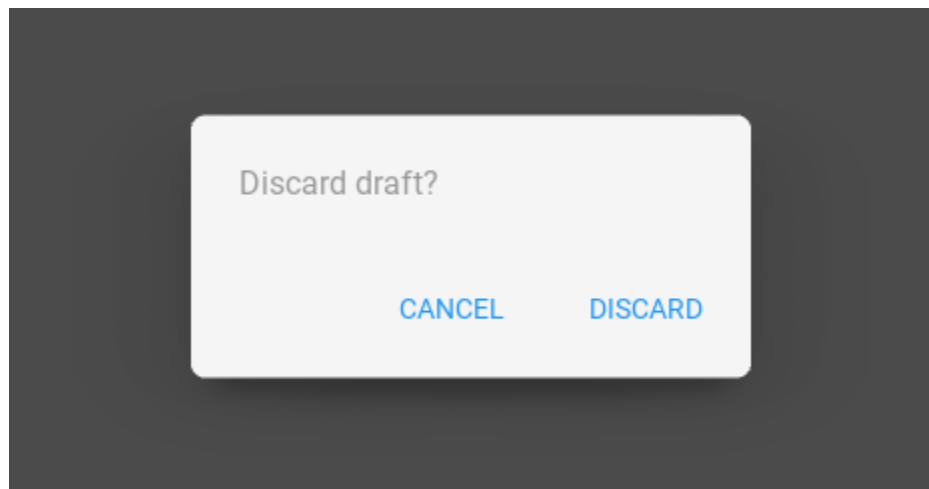
(continued from previous page)

```
class Example(MDApp):
    dialog = None

    def build(self):
        return Builder.load_string(KV)

    def show_alert_dialog(self):
        if not self.dialog:
            self.dialog = MDDialog(
                text="Discard draft?",
                buttons=[
                    MDFlatButton(
                        text="CANCEL",
                        theme_text_color="Custom",
                        text_color=self.theme_cls.primary_color,
                    ),
                    MDFlatButton(
                        text="DISCARD",
                        theme_text_color="Custom",
                        text_color=self.theme_cls.primary_color,
                    ),
                ],
            )
            self.dialog.open()
```

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



API - kivymd.uix.dialog.dialog**class** kivymd.uix.dialog.dialog.**BaseDialog**(**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.

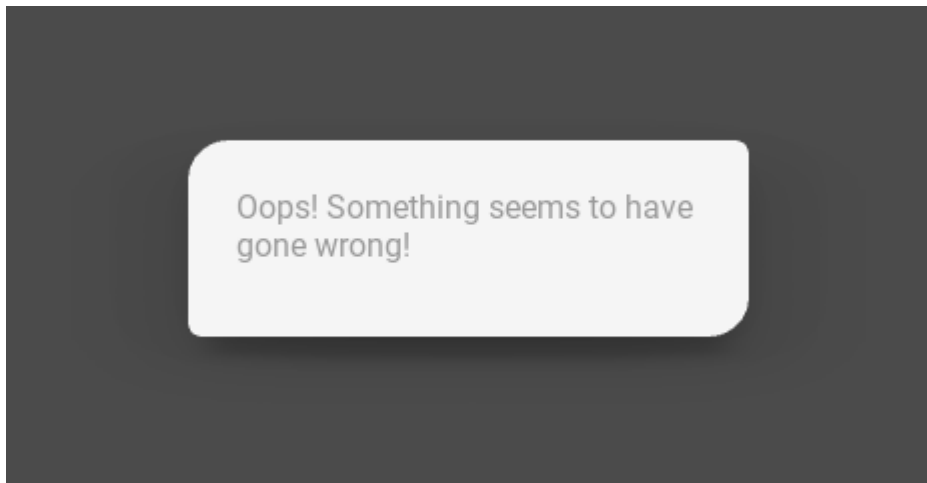
Changed in version 1.11.0: Added events *on_pre_open* and *on_pre_dismiss*.

Changed in version 2.0.0: Added property 'overlay_color'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.**radius**

Dialog corners rounding value.

```
[...]
self.dialog = MDDialog(
    text="Oops! Something seems to have gone wrong!",
    radius=[20, 7, 20, 7],
)
[...]
```

*radius* is an `ListProperty` and defaults to `[7, 7, 7, 7]`.**class** kivymd.uix.dialog.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*.

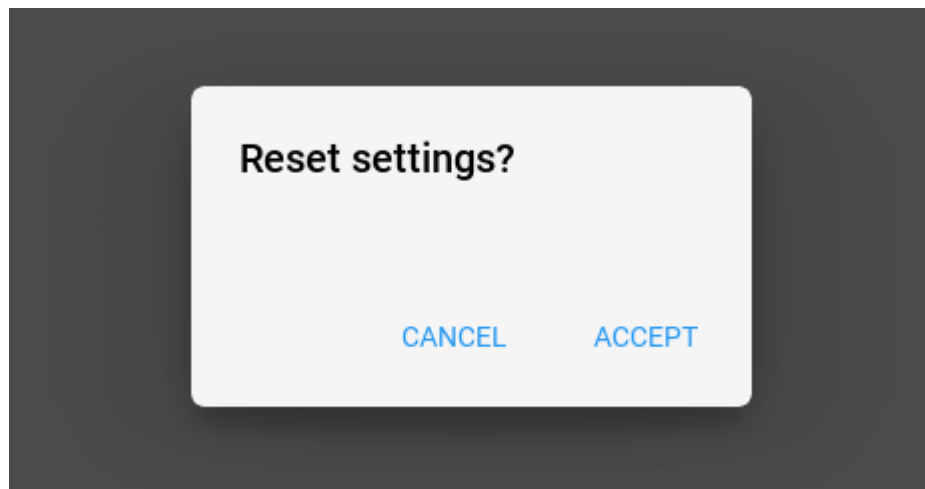
Changed in version 2.0.0: Added property 'overlay_color'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

title

Title dialog.

```
[...]
self.dialog = MDDialog(
    title="Reset settings?",
    buttons=[
        MDFlatButton(
            text="CANCEL",
            theme_text_color="Custom",
            text_color=self.theme_cls.primary_color,
        ),
        MDFlatButton(
            text="ACCEPT",
            theme_text_color="Custom",
            text_color=self.theme_cls.primary_color,
        ),
    ],
)
[...]
```

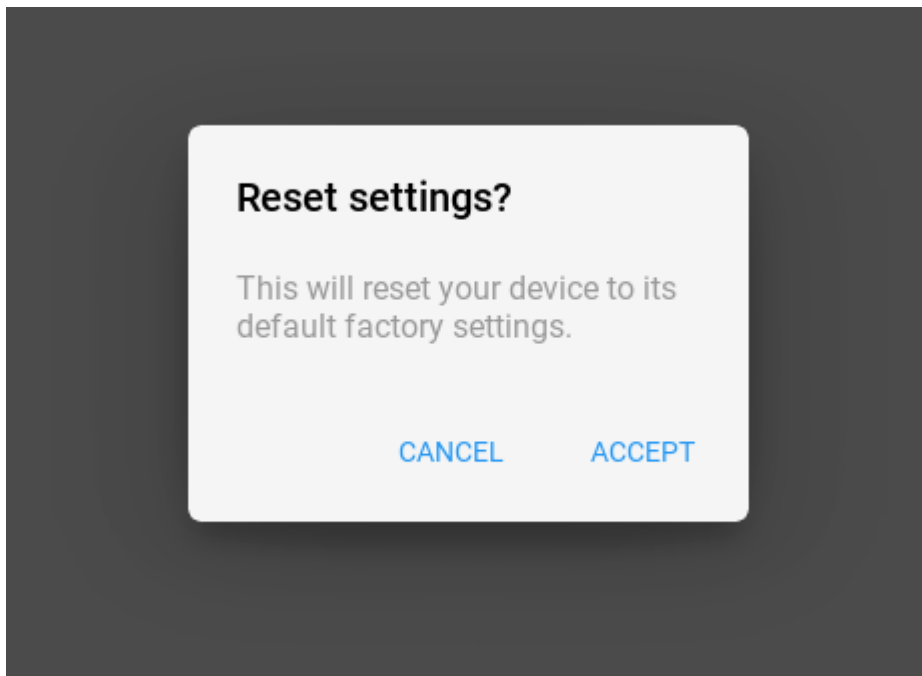


title is an [StringProperty](#) and defaults to ''.

text

Text dialog.

```
[...]
self.dialog = MDDialog(
    title="Reset settings?",
    text="This will reset your device to its default factory settings.",
    buttons=[
        MDFlatButton(
            text="CANCEL",
            theme_text_color="Custom",
            text_color=self.theme_cls.primary_color,
        ),
        MDFlatButton(
            text="ACCEPT",
            theme_text_color="Custom",
            text_color=self.theme_cls.primary_color,
        ),
    ],
)
[...]
```



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

buttons

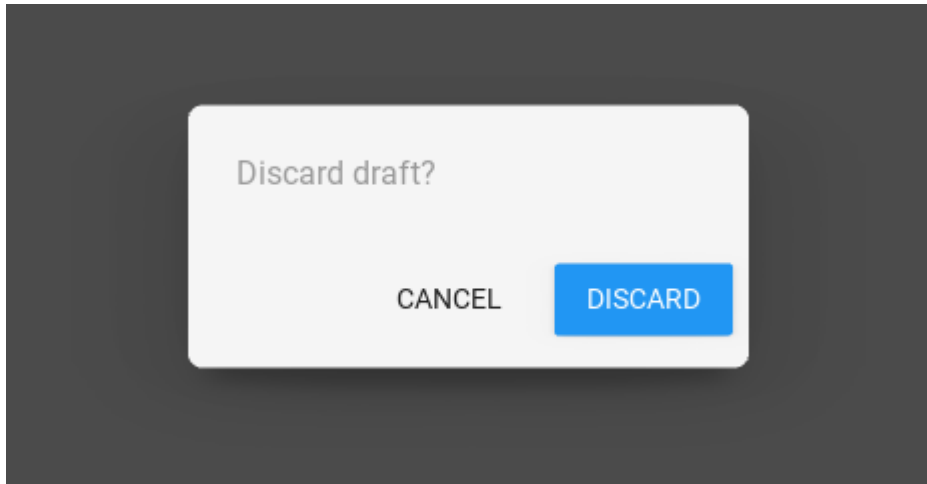
List of button objects for dialog. Objects must be inherited from `BaseButton` class.

```
[...]
self.dialog = MDDialog(
    text="Discard draft?",
    buttons=[
        MDFlatButton(text="CANCEL"), MDRaisedButton(text="DISCARD"),
```

(continues on next page)

(continued from previous page)

```
    ],
)
[...]
```



`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()
```

(continues on next page)

(continued from previous page)

```

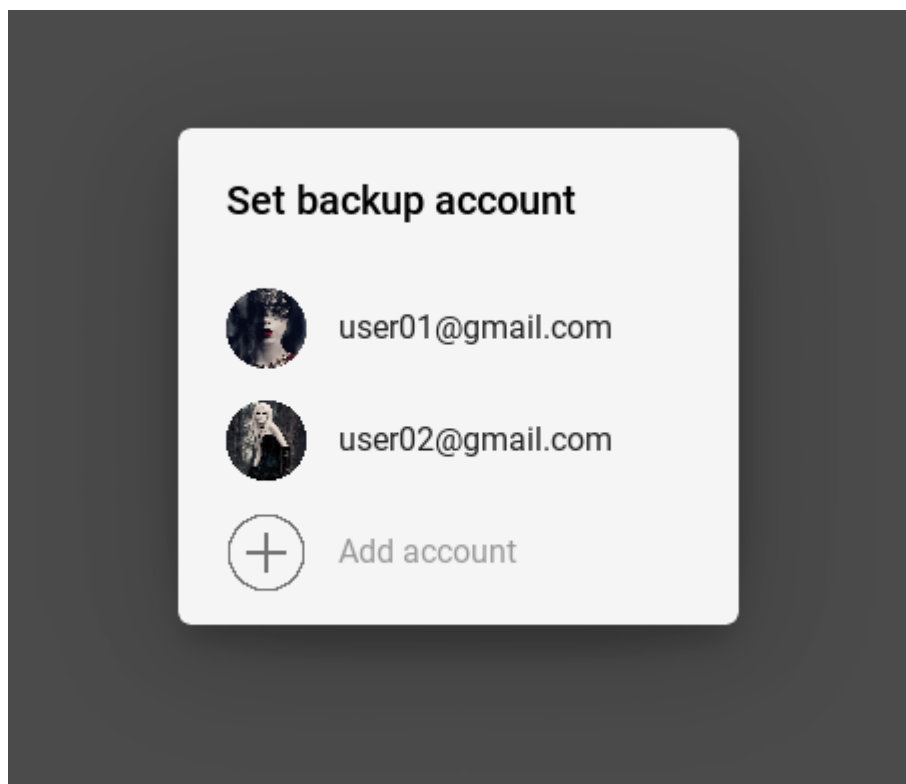
class Example(MDApp):
    dialog = None

    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

```

(continues on next page)

(continued from previous page)

```

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

    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"),

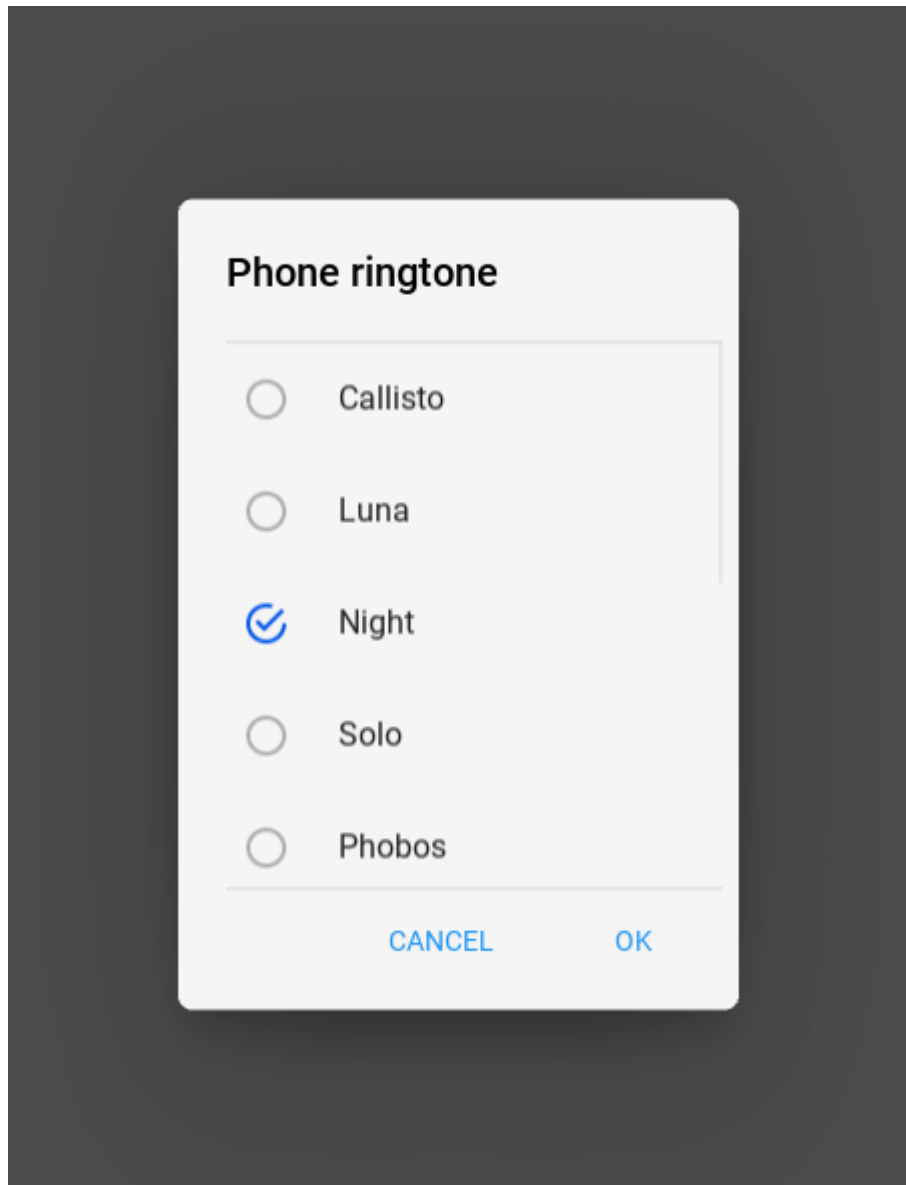
```

(continues on next page)

(continued from previous page)

```
        ItemConfirm(text="Allergio"),
        ItemConfirm(text="Magic"),
        ItemConfirm(text="Tic-tac"),
    ],
    buttons=[
        MDFlatButton(
            text="CANCEL",
            theme_text_color="Custom",
            text_color=self.theme_cls.primary_color,
        ),
        MDFlatButton(
            text="OK",
            theme_text_color="Custom",
            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:

    MDFlatButton:
        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",
                        theme_text_color="Custom",
                        text_color=self.theme_cls.primary_color,
                    ),
                    MDFlatButton(
                        text="OK",

```

(continues on next page)

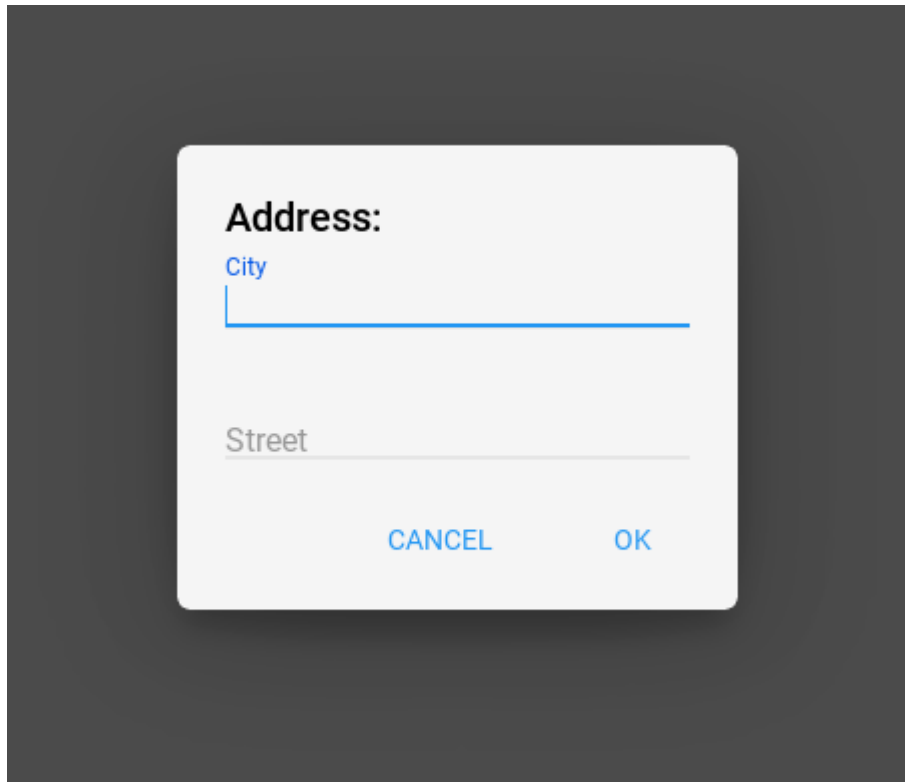
(continued from previous page)

```

        theme_text_color="Custom",
        text_color=self.theme_cls.primary_color,
    ),
    ],
)
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)

update_items(self, items: list)

on_open(self)

default open event handler.

get_normal_height(self)

edit_padding_for_item(self, instance_item)

```
create_items(self)

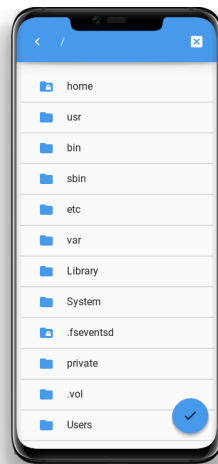
create_buttons(self)
```

2.3.48 FileManager

A simple manager for selecting directories and files.

Usage

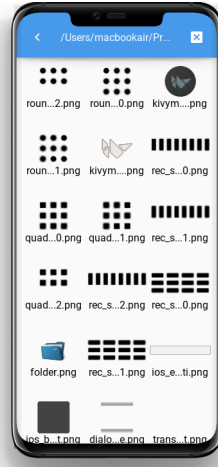
```
path = '/' # path to the directory that will be opened in the file manager
file_manager = MDFileManager(
    exit_manager=self.exit_manager, # function called when the user reaches directory
    ↪tree root
    select_path=self.select_path, # function called when selecting a file/directory
)
file_manager.show(path)
```



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 MDFileManager
from kivymd.toast import toast

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

    MDTopAppBar:
        title: "MDFileManager"
        left_action_items: [['menu', lambda x: None]]
        elevation: 10

    MDFloatLayout:

        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)
```

(continues on next page)

(continued from previous page)

```

Window.bind(on_keyboard=self.events)
self.manager_open = False
self.file_manager = MDFileManager(
    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;
    :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.filemanager

class kivymd.uix.filemanager.filemanager.MDFileManager(**kwargs)

Relative layout class. For more information, see in the [RelativeLayout](#) class documentation.

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: str)**

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: str)**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: str, widget: Union[BodyManagerWithPreview, Factory.BodyManager])

Called by tap on the name of the directory or file.

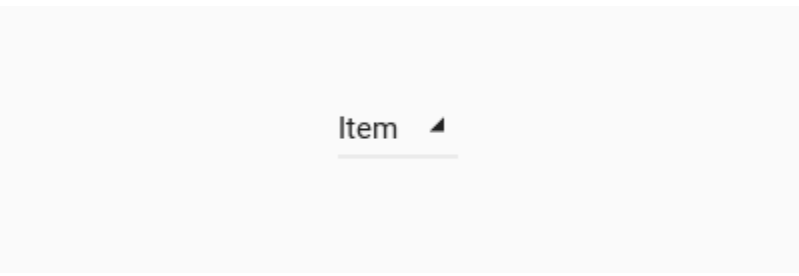
back(self)

Returning to the branch down in the directory tree.

select_directory_on_press_button(self, *args)

Called when a click on a floating button.

2.3.49 DropdownItem



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.dropdownitem`

class `kivymd.uix.dropdownitem.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 behind 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 its content in the canvas in the correct order, avoiding some visual errors.

FakeCircularElevationBehavior will load prefabricated textures to optimize loading times.

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_drop_down_item*, *text_item*: *str*)

set_item(*self*, *name_item*: *str*)

Sets new text for an item.

2.3.50 Transition

A set of classes for implementing transitions between application screens.

New in version 1.0.0.

Changing transitions

You have multiple transitions available by default, such as:

- ***MDFadeSlideTransition***

state one: the new screen closes the previous screen by lifting from the bottom of the screen and changing from transparent to non-transparent;

state two: the current screen goes down to the bottom of the screen, passing from a non-transparent state to a transparent one, thus opening the previous screen;

Note: You cannot control the direction of a slide using the *direction* attribute.

API - `kivymd.uix.transition.transition`

`class kivymd.uix.transition.transition.MDTransitionBase`

TransitionBase is used to animate 2 screens within the ScreenManager. This class acts as a base for other implementations like the SlideTransition and SwapTransition.

Events

`on_progress`: Transition object, progression float
Fired during the animation of the transition.

`on_complete`: Transition object
Fired when the transition is finished.

`hero_widget`

`hero_from_widget`

`start`(*self*, *instance_screen_manager*: [MDScreenManager](#))

(internal) Starts the transition. This is automatically called by the ScreenManager.

`animated_hero_in`(*self*)

`animated_hero_out`(*self*)

`on_complete`(*self*)

`class kivymd.uix.transition.transition.MDSwapTransition(**kwargs)`

Swap transition that looks like iOS transition when a new window appears on the screen.

`class kivymd.uix.transition.transition.MDSlideTransition`

Slide Transition, can be used to show a new screen from any direction: left, right, up or down.

`class kivymd.uix.transition.transition.MDFadeSlideTransition`

Slide Transition, can be used to show a new screen from any direction: left, right, up or down.

`start`(*self*, *instance_screen_manager*: [MDScreenManager](#))

(internal) Starts the transition. This is automatically called by the ScreenManager.

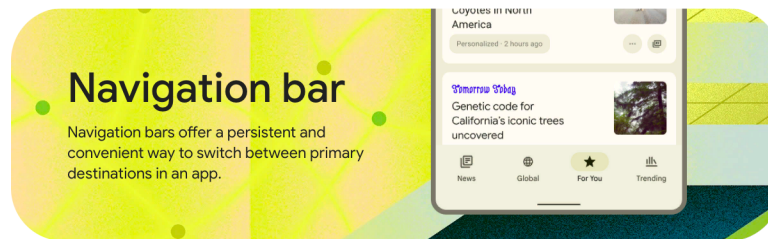
`on_progress`(*self*, *progression*: *float*)

2.3.51 BottomNavigation

See also:

[Material Design 2 spec, Bottom navigation](#) and [Material Design 3 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 kivy.lang import Builder

from kivymd.app import MDApp

class Test(MDApp):

    def build(self):
        self.theme_cls.material_style = "M3"
        return Builder.load_string(
            '''
MDScreen:

    MDBottomNavigation:
        panel_color: "#eeeeea"
        selected_color_background: "#97ecf8"
        text_color_active: 0, 0, 0, 1

        MDBottomNavigationItem:
            name: 'screen 1'
            text: 'Mail'
            icon: 'gmail'
            badge_icon: "numeric-10"

            MDLabel:
                text: 'Mail'
                halign: 'center'

        MDBottomNavigationItem:
            name: 'screen 2'
            text: 'Discord'
            icon: 'discord'
            badge_icon: "numeric-5"

            MDLabel:
                text: 'Discord'
                halign: 'center'

        MDBottomNavigationItem:
            name: 'screen 3'
            text: 'LinkedIN'
            icon: 'linkedin'

            MDLabel:
                text: 'LinkedIN'
                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",  
)
```

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?

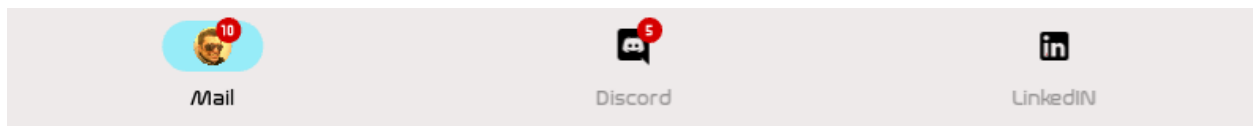
Use method `switch_tab` which takes as argument the name of the tab you want to switch to.

Use custom icon

MDBottomNavigation:

MDBottomNavigationItem:

```
icon: "icon.png"
```



API - kivymd.uix.bottomnavigation.bottomnavigation

class kivymd.uix.bottomnavigation.bottomnavigation.MDTab(*args, **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'*.

badge_icon

Tab header badge icon.

New in version 1.0.0.

badge_icon is an *StringProperty* and defaults to *''*.

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.bottomnavigation.MDBottomNavigationItem(*args, **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)

Called when clicking on a panel item.

on_disabled(self, instance_bottom_navigation_item, disabled_value: *bool*)

This function hides the shadow when the widget is disabled. It sets the shadow to 0.

on_leave(self, *args)

class kivymd.uix.bottomnavigation.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 `ColorProperty` and defaults to `None`.

tabs

class `kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation(*args, **kwargs)`

A bottom navigation that is implemented by delegating all items to a `ScreenManager`.

Events**`on_switch_tabs`**

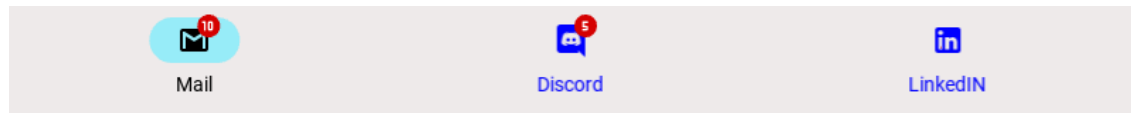
Called when switching tabs. Returns the object of the tab to be opened.

New in version 1.0.0.

text_color_normal

Text color of the label when it is not selected.

```
MDBottomNavigation:
    text_color_normal: 1, 0, 1, 1
```

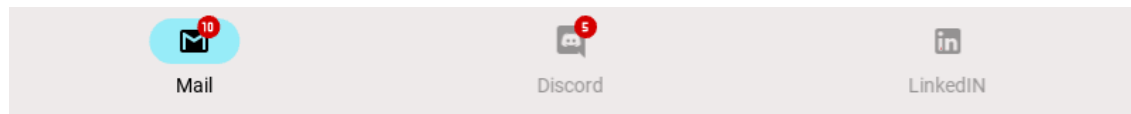


`text_color_normal` is an `ColorProperty` and defaults to `[1, 1, 1, 1]`.

text_color_active

Text color of the label when it is selected.

```
MDBottomNavigation:
    text_color_active: 0, 0, 0, 1
```

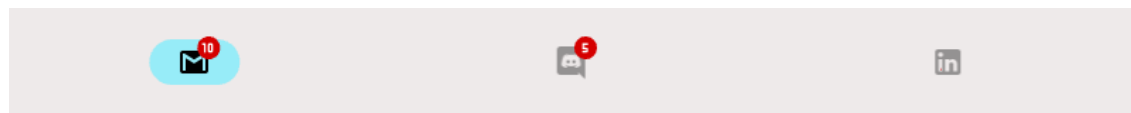


`text_color_active` is an `ColorProperty` and defaults to `[1, 1, 1, 1]`.

use_text

Use text for `MDBottomNavigationItem` or not. If `True`, the `MDBottomNavigation` panel height will be reduced by the text height.

New in version 1.0.0.



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

selected_color_background

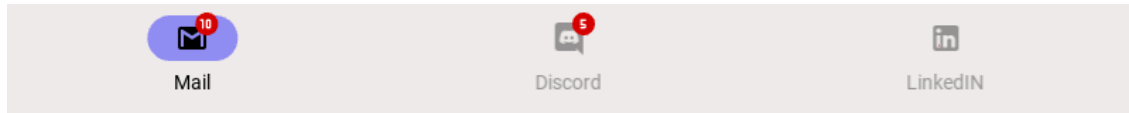
The background color of the highlighted item when using Material Design v3.

New in version 1.0.0.

```

MDBottomNavigation:
    selected_color_background: 0, 0, 1, .4

```



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

font_name

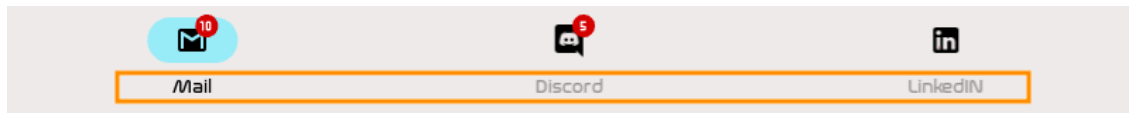
Font name of the label.

New in version 1.0.0.

```

MDBottomNavigation:
    font_name: "path/to/font.ttf"

```



`font_name` is an `StringProperty` and defaults to `'Roboto'`.

first_widget

`first_widget` is an `MDBottomNavigationItem` and defaults to `None`.

tab_header

`tab_header` is an `MDBottomNavigationHeader` and defaults to `None`.

setBars_color

If `True` the background color of the navigation bar will be set automatically according to the current color of the toolbar.

New in version 1.0.0.

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

widget_index

`set_status_bar_color(self, interval: Union[int, float])`

`switch_tab(self, name_tab)`

Switching the tab by name.

`refresh_tabs(self, *args)`

Refresh all tabs.

`on_font_name(self, instance_bottom_navigation, font_name: str)`

`on_selected_color_background(self, instance_bottom_navigation, color: list)`

`on_use_text(self, instance_bottom_navigation, use_text_value: bool)`

`on_text_color_normal(self, instance_bottom_navigation, color: list)`

`on_text_color_active(self, instance_bottom_navigation, color: list)`

on_switch_tabs(self, bottom_navigation_item, name_tab: str)

Called when switching tabs. Returns the object of the tab to be opened.

on_size(self, *args)

on_resize(self, instance: Union[WindowSDL, None] = None, width: Union[int, None] = None, do_again: bool = 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)
```

2.3.52 Swiper

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:

    MDTopAppBar:
        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:

        MySwiper:

        MySwiper:

        MySwiper:
'''

```

(continues on next page)

(continued from previous page)

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

Main().run()
```

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

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

MDSwipe provides the following events for use:

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

```
MDSwipe:
    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>

<MySwipe@MDSwipeItem>

    RelativeLayout:
```

(continues on next page)

(continued from previous page)

```

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

        MDBoxLayout:
            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:

    MDTopAppBar:
        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:

    MySwiper:

    MySwiper:

    MySwiper:

    MySwiper:

    ...

class Main(MDApp):
    def build(self):

```

(continues on next page)

(continued from previous page)

```
        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.swiper

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

class kivymd.uix.swiper.swiper.**MDSwiper**(*args, **kwargs)
ScrollView class. For more information, see in the `ScrollView` class documentation.

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

Whether 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_pre_swipe(*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.

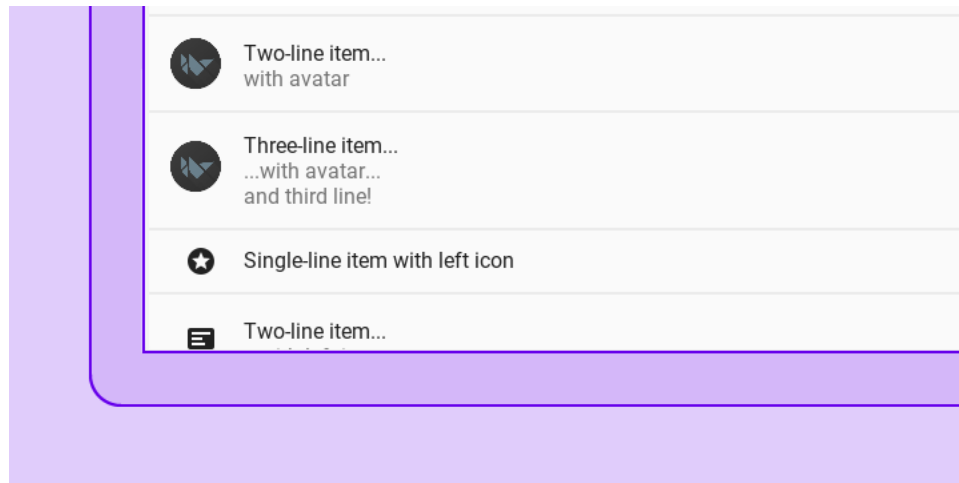
See [on_touch_down\(\)](#) for more information.

2.3.53 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)
```

(continues on next page)

(continued from previous page)

```

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()

```

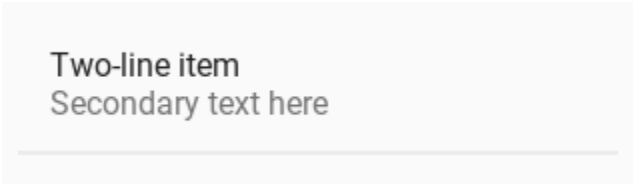
OneLineListItem

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



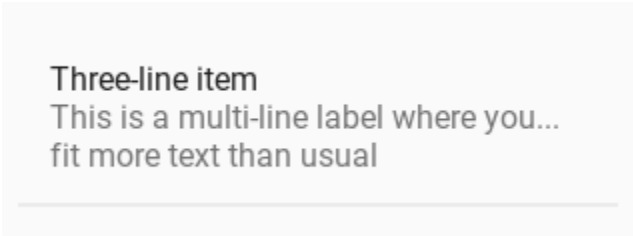
TwoLineListItem

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



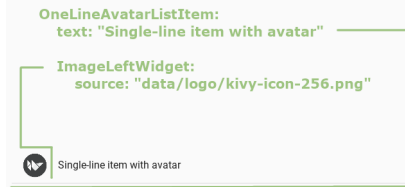
ThreeLineListItem

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



OneLineAvatarListItem

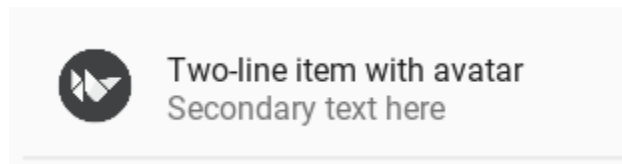
```
OneLineAvatarListItem:  
    text: "Single-line item with avatar"  
  
    ImageLeftWidget:  
        source: "data/logo/kivy-icon-256.png"
```



TwoLineAvatarListItem

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

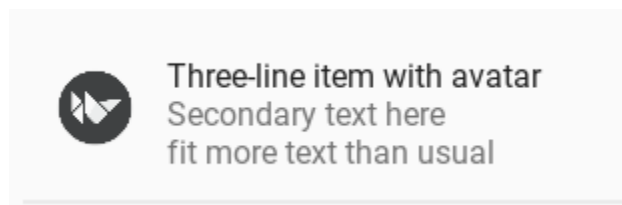
    ImageLeftWidget:
        source: "data/logo/kivy-icon-256.png"
```



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"
```



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"
```



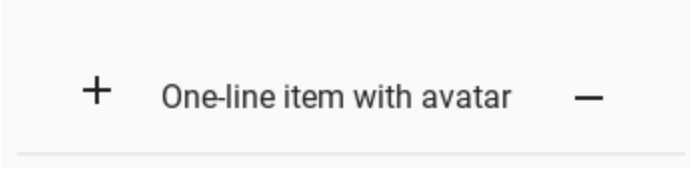
Three-line item with avatar
Secondary text here
fit more text than usual

OneLineAvatarIconListItem

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

    IconLeftWidget:
        icon: "plus"

    IconRightWidget:
        icon: "minus"
```

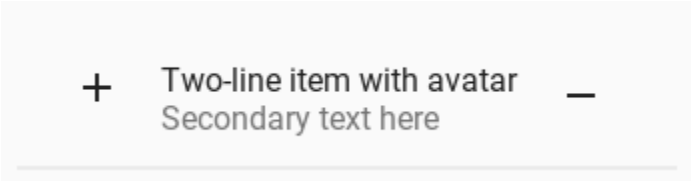
+ One-line item with avatar —

TwoLineAvatarIconListItem

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

    IconLeftWidget:
        icon: "plus"

    IconRightWidget:
        icon: "minus"
```



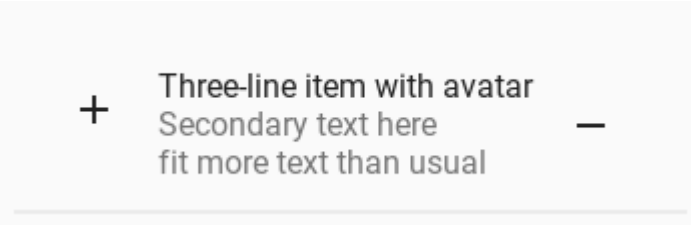
+ Two-line item with avatar
Secondary text here —

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

KV = '''
<ListItemWithCheckbox>:

    IconLeftWidget:
        icon: root.icon

    RightCheckbox:

MDBoxLayout:

    ScrollView:

        MDList:
            id: scroll
'''

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

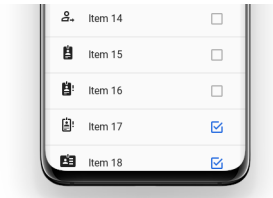
    icon = StringProperty("android")

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

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
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"

    YourContainer:
        id: container

        MDIconButton:
            icon: "minus"

        MDIconButton:
            icon: "plus"
'''

class YourContainer(IRightBodyTouch, MDBoxLayout):
    adaptive_width = True

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

MainApp().run()

```



One-line item with avatar



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.list

class kivymd.uix.list.list.**MDList**(*args, **kwargs)

ListItem container. Best used in conjunction with a `kivy.uix.ScrollView`.

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())
```

(continues on next page)

(continued from previous page)

```
>>> 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.list.BaseListItem(*args, **kwargs)

Base class to all ListItems. Not supposed to be instantiated on its own.

text

Text shown in the first line.

text is a [StringProperty](#) and defaults to ''.

text_color

Text color in rgba format used if *theme_text_color* is set to 'Custom'.

text_color is a [ColorProperty](#) and defaults to None.

font_style

Text font style. See `kivyMD.font_definitions.py`.

font_style is a [StringProperty](#) and defaults to 'Subtitle1'.

theme_text_color

Theme text color in rgba format for primary text.

theme_text_color is a [StringProperty](#) and defaults to 'Primary'.

secondary_text

Text shown in the second line.

secondary_text is a [StringProperty](#) and defaults to ''.

tertiary_text

The text is displayed on the third line.

tertiary_text is a [StringProperty](#) and defaults to ''.

secondary_text_color

Text color in rgba format used for secondary text if *secondary_theme_text_color* is set to 'Custom'.

secondary_text_color is a [ColorProperty](#) and defaults to None.

tertiary_text_color

Text color in rgba format used for tertiary text if *tertiary_theme_text_color* is set to 'Custom'.

tertiary_text_color is a [ColorProperty](#) and defaults to None.

secondary_theme_text_color

Theme text color for secondary text.

`secondary_theme_text_color` is a `StringProperty` and defaults to `'Secondary'`.

tertiary_theme_text_color

Theme text color for tertiary text.

`tertiary_theme_text_color` is a `StringProperty` and defaults to `'Secondary'`.

secondary_font_style

Font style for secondary line. See `kivymd.font_definitions.py`.

`secondary_font_style` is a `StringProperty` and defaults to `'Body1'`.

tertiary_font_style

Font style for tertiary line. See `kivymd.font_definitions.py`.

`tertiary_font_style` is a `StringProperty` and defaults to `'Body1'`.

divider

Divider mode. Available options are: `'Full'`, `'Inset'` and default to `'Full'`.

`divider` is a `OptionProperty` and defaults to `'Full'`.

divider_color

Divider color.

New in version 1.0.0.

`divider_color` is a `ColorProperty` and defaults to `None`.

bg_color

Background color for menu item.

`bg_color` is a `ColorProperty` and defaults to `None`.

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]`.

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*, **args*)

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.

propagate_touch_to_touchable_widgets(*self*, *touch*, *touch_event*, **args*)

add_widget(*self*, *widget*)

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.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.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.list.OneLineListItem(*args, **kwargs)

A one line list item.

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

A two line list item.

class kivymd.uix.list.list.ThreeLineListItem(*args, **kwargs)

A three line list item.

class kivymd.uix.list.list.OneLineAvatarListItem(*args, **kwargs)

Base class to all ListItems. Not supposed to be instantiated on its own.

class kivymd.uix.list.list.TwoLineAvatarListItem(*args, **kwargs)

Base class to all ListItems. Not supposed to be instantiated on its own.

class kivymd.uix.list.list.ThreeLineAvatarListItem(*args, **kwargs)

A three line list item.

class kivymd.uix.list.list.OneLineIconListItem(*args, **kwargs)

A one line list item.

class kivymd.uix.list.list.TwoLineIconListItem(*args, **kwargs)

A one line list item.

class kivymd.uix.list.list.ThreeLineIconListItem(*args, **kwargs)

A three line list item.

class kivymd.uix.list.list.OneLineRightIconListItem(*args, **kwargs)

A one line list item.

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

A one line list item.

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

A three line list item.

class kivymd.uix.list.list.OneLineAvatarIconListItem(*args, **kwargs)

Base class to all ListItems. Not supposed to be instantiated on its own.

class kivymd.uix.list.list.TwoLineAvatarIconListItem(*args, **kwargs)

Base class to all ListItems. Not supposed to be instantiated on its own.

class kivymd.uix.list.list.ThreeLineAvatarIconListItem(*args, **kwargs)

A three line list item.

class kivymd.uix.list.list.ImageLeftWidget(**kwargs)

Class implements a circular ripple effect.

class kivymd.uix.list.list.ImageLeftWidgetWithoutTouch(**kwargs)

New in version 1.0.0.

class kivymd.uix.list.list.ImageRightWidget(**kwargs)

Class implements a circular ripple effect.


```
class kivymd.uix.list.list.ImageRightWidgetWithoutTouch(**kwargs)
```

New in version 1.0.0.

```
class kivymd.uix.list.list.IconRightWidget(**kwargs)
```

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

pos_hint

```
class kivymd.uix.list.list.IconRightWidgetWithoutTouch(**kwargs)
```

New in version 1.0.0.

pos_hint

```
class kivymd.uix.list.list.IconLeftWidget(**kwargs)
```

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

pos_hint

```
class kivymd.uix.list.list.IconLeftWidgetWithoutTouch(**kwargs)
```

New in version 1.0.0.

pos_hint

```
class kivymd.uix.list.list.CheckboxLeftWidget(**kwargs)
```

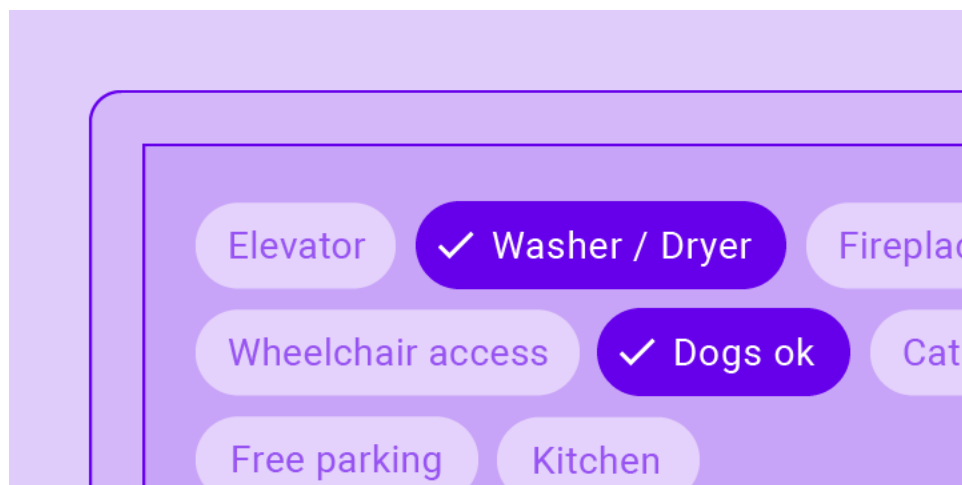
Same as ILeftBody, but allows the widget to receive touch events instead of triggering the ListItem's ripple effect.

2.3.54 Chip

See also:

[Material Design spec, Chips](#)

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



Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

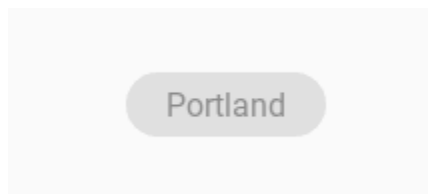
KV = '''
MDScreen:

    MDChip:
        text: "Portland"
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.on_release_chip(self)
'''

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

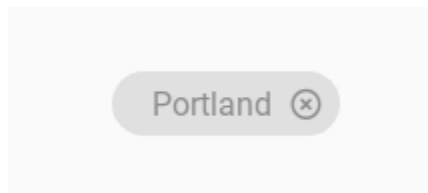
    def on_release_chip(self, instance_check):
        print(instance_check)

Test().run()
```



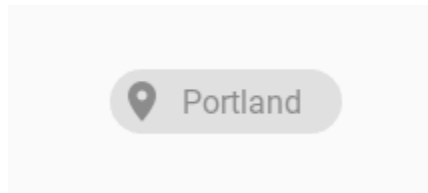
Use with right icon

```
MDChip:
    text: "Portland"
    icon_right: "close-circle-outline"
```



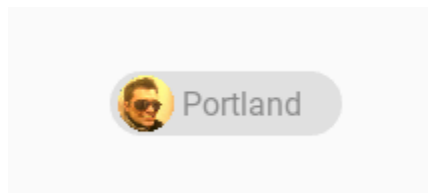
Use with left icon

```
MDChip:  
    text: "Portland"  
    icon_left: "map-marker"
```



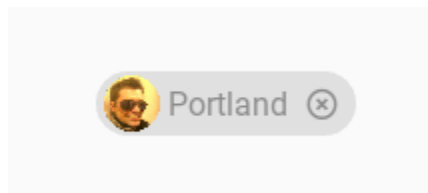
Use with custom left icon

```
MDChip:  
    text: "Portland"  
    icon_left: "avatar.png"
```



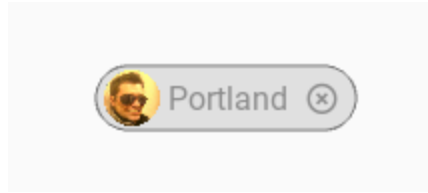
Use with left and right icon

```
MDChip:  
    text: "Portland"  
    icon_left: "avatar.png"  
    icon_right: "close-circle-outline"
```



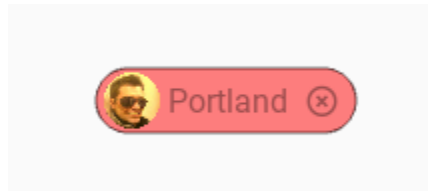
Use with outline

```
MDChip:  
    text: "Portland"  
    icon_left: "avatar.png"  
    icon_right: "close-circle-outline"  
    line_color: app.theme_cls.disabled_hint_text_color
```



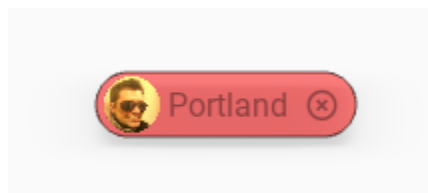
Use with custom color

```
MDChip:
    text: "Portland"
    icon_left: "avatar.png"
    icon_right: "close-circle-outline"
    line_color: app.theme_cls.disabled_hint_text_color
    md_bg_color: 1, 0, 0, .5
```



Use with elevation

```
MDChip:
    text: "Portland"
    icon_left: "avatar.png"
    icon_right: "close-circle-outline"
    line_color: app.theme_cls.disabled_hint_text_color
    md_bg_color: 1, 0, 0, .5
    elevation: 12
```



Behavior

Long press on the chip, it will be marked. When you click on the marked chip, the mark will be removed:

Examples

Multiple choose

Selecting a single choice chip automatically deselects all other chips in the set.

```
from kivy.animation import Animation
from kivy.lang import Builder

from kivymd.uix.screen import MDScreen
from kivymd.uix.chip import MDChip
from kivymd.app import MDApp

KV = '''
<MyScreen>

    MDBoxLayout:
        orientation: "vertical"
        adaptive_size: True
        spacing: "12dp"
        padding: "56dp"
        pos_hint: {"center_x": .5, "center_y": .5}

        MDLabel:
            text: "Multiple choice"
            bold: True
            font_style: "H5"
            adaptive_size: True

        MDBoxLayout:
            id: chip_box
            adaptive_size: True
            spacing: "8dp"

            MyChip:
                text: "Elevator"
                on_press: if self.active: root.removes_marks_all_chips()

            MyChip:
                text: "Washer / Dryer"
                on_press: if self.active: root.removes_marks_all_chips()

            MyChip:
                text: "Fireplace"
                on_press: if self.active: root.removes_marks_all_chips()

ScreenManager:

    MyScreen:
'''
```

(continues on next page)

(continued from previous page)

```

class MyChip(MDChip):
    icon_check_color = (0, 0, 0, 1)
    text_color = (0, 0, 0, 0.5)
    _no_ripple_effect = True

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.bind(active=self.set_chip_bg_color)
        self.bind(active=self.set_chip_text_color)

    def set_chip_bg_color(self, instance_chip, active_value: int):
        """
        Will be called every time the chip is activated/deactivated.
        Sets the background color of the chip.
        """

        self.md_bg_color = (
            (0, 0, 0, 0.4)
            if active_value
            else (
                self.theme_cls.bg_darkest
                if self.theme_cls.theme_style == "Light"
                else (
                    self.theme_cls.bg_light
                    if not self.disabled
                    else self.theme_cls.disabled_hint_text_color
                )
            )
        )

    def set_chip_text_color(self, instance_chip, active_value: int):
        Animation(
            color=(0, 0, 0, 1) if active_value else (0, 0, 0, 0.5), d=0.2
        ).start(self.ids.label)

class MyScreen(MDScreen):
    def removes_marks_all_chips(self):
        for instance_chip in self.ids.chip_box.children:
            if instance_chip.active:
                instance_chip.active = False

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

Test().run()

```

Only choose

Only one chip will be selected.

```

KV = '''
<MyScreen>

    [...]

    MDBoxLayout:
        id: chip_box
        adaptive_size: True
        spacing: "8dp"

        MyChip:
            text: "Elevator"
            on_active: if self.active: root.removes_marks_all_chips(self)

        MyChip:
            text: "Washer / Dryer"
            on_active: if self.active: root.removes_marks_all_chips(self)

        MyChip:
            text: "Fireplace"
            on_active: if self.active: root.removes_marks_all_chips(self)

    [...]
'''

class MyScreen(MDScreen):
    def removes_marks_all_chips(self, selected_instance_chip):
        for instance_chip in self.ids.chip_box.children:
            if instance_chip != selected_instance_chip:
                instance_chip.active = False

```

API - kivymd.uix.chip.chip

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

Class implements a rectangular ripple effect.

text

Chip text.

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

icon_left

Chip left icon.

New in version 1.0.0.

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

icon_right

Chip right icon.

New in version 1.0.0.

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

text_color

Chip's text color in `rgba` format.

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

icon_right_color

Chip's right icon color in `rgba` format.

New in version 1.0.0.

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

icon_left_color

Chip's left icon color in `rgba` format.

New in version 1.0.0.

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

icon_check_color

Chip's check icon color in `rgba` format.

New in version 1.0.0.

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

active

Whether the check is marked or not.

New in version 1.0.0.

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

on_long_touch(*self*, *args)

Called when the widget is pressed for a long time.

on_active(*self*, instance_check, active_value: *bool*)**do_animation_check**(*self*, md_bg_color: *list*, scale_value: *int*)**on_press**(*self*, *args)

2.4 Controllers

2.4.1 WindowController

New in version 1.0.0.

Modules and classes that implement useful methods for getting information about the state of the current application window.

Controlling the resizing direction of the application window

```
# When resizing the application window, the direction of change will be
# printed - 'left' or 'right'.

from kivymd.app import MDApp
from kivymd.uix.controllers import WindowController
from kivymd.uix.screen import MDScreen

class MyScreen(MDScreen, WindowController):
    def on_width(self, *args):
        print(self.get_window_width_resizing_direction())

class Test(MDApp):
    def build(self):
        return MyScreen()

Test().run()
```

API - `kivymd.uix.controllers.windowcontroller`

```
class kivymd.uix.controllers.windowcontroller.WindowController

    on_size(self, instance, size: list)
        Called when the application screen size changes.

    get_real_device_type(self)
        Returns the device type - 'mobile', 'tablet' or 'desktop'.

    get_window_width_resizing_direction(self)
        Return window width resizing direction - 'left' or 'right'.
```

2.5 Behaviors

2.5.1 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.

2.5.2 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}

        MyToggleButton:
            text: "Show ads"
            group: "x"

        MyToggleButton:
            text: "Do not show ads"
            group: "x"
```

(continues on next page)

(continued from previous page)

```
        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]`.

2.5.3 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 MDBoxLayout
from kivymd.theming import ThemableBehavior

KV = '''
Screen

    MDBoxLayout:
        id: box
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: .8, .8
        md_bg_color: app.theme_cls.bg_darkest
'''

class HoverItem(MDBoxLayout, 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.

2.5.4 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**(**kwarg)

Common base class for rectangular and circular elevation behavior.

background

Background image path.

background is a [StringProperty](#) and defaults to *None*.**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 [ColorProperty](#) and defaults to *[1, 1, 1, 0]*.**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]*.

line_width

Border of the specified width will be used to border the widget.

New in version 1.0.0.

`line_width` is an `NumericProperty` and defaults to `1`.

angle**background_origin**

`update_background_origin(self, instance_md_widget, pos: List[float])`

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]`.

2.5.5 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 = '''
MDScreen:

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

(continues on next page)

(continued from previous page)

```

        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):
        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 = '''
MDScreen:

    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):
        return Builder.load_string(KV)

Example().run()

```

API - kivymd.uix.behaviors.ripple_behavior

class kivymd.uix.behaviors.ripple_behavior.CommonRipple

Base class for ripple effect.

ripple_rad_default

The starting value of the radius of the ripple effect.

```
CircularRippleButton:
    ripple_rad_default: 100
```

ripple_rad_default is an *NumericProperty* and defaults to 1.

ripple_color

Ripple color in (r, g, b, a) format.

```
CircularRippleButton:
    ripple_color: app.theme_cls.primary_color
```

ripple_color is an *ColorProperty* and defaults to *None*.

ripple_alpha

Alpha channel values for ripple effect.

```
CircularRippleButton:
    ripple_alpha: .9
    ripple_color: app.theme_cls.primary_color
```

ripple_alpha is an *NumericProperty* and defaults to 0.5.

ripple_scale

Ripple effect scale.

```
CircularRippleButton:
    ripple_scale: .5
```

```
CircularRippleButton:
    ripple_scale: 1
```

ripple_scale is an *NumericProperty* and defaults to *None*.

ripple_duration_in_fast

Ripple duration when touching to widget.

```
CircularRippleButton:  
    ripple_duration_in_fast: .1
```

ripple_duration_in_fast is an `NumericProperty` and defaults to *0.3*.

ripple_duration_in_slow

Ripple duration when long touching to widget.

```
CircularRippleButton:  
    ripple_duration_in_slow: 5
```

ripple_duration_in_slow is an `NumericProperty` and defaults to *2*.

ripple_duration_out

The duration of the disappearance of the wave effect.

```
CircularRippleButton:  
    ripple_duration_out: 5
```

ripple_duration_out is an `NumericProperty` and defaults to *0.3*.

ripple_canvas_after

The ripple effect is drawn above/below the content.

New in version 1.0.0.

```
MDIconButton:  
    ripple_canvas_after: True  
    icon: "android"  
    ripple_alpha: .8  
    ripple_color: app.theme_cls.primary_color  
    icon_size: "100sp"
```

```
MDIconButton:  
    ripple_canvas_after: False  
    icon: "android"  
    ripple_alpha: .8  
    ripple_color: app.theme_cls.primary_color  
    icon_size: "100sp"
```

ripple_canvas_after is an `BooleanProperty` and defaults to *True*.

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)****call_ripple_animation_methods(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)

2.5.6 Declarative

New in version 1.0.0.

As you already know, the Kivy framework provides the best/simplest/modern UI creation tool that allows you to separate the logic of your application from the description of the properties of widgets/GUI components. This tool is named *KV Language*.

But in addition to creating a user interface using the KV Language Kivy allows you to create user interface elements directly in the Python code. And if you've ever created a user interface in Python code, you know how ugly it looks. Even in the simplest user interface design, which was created using Python code it is impossible to trace the widget tree, because in Python code you build the user interface in an imperative style.

Imperative style

```
from kivymd.app import MDApp
from kivymd.uix.bottomnavigation import MDBottomNavigation, MDBottomNavigationItem
from kivymd.uix.label import MDLabel
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
        screen = MDScreen()
        bottom_navigation = MDBottomNavigation(
            panel_color="#eeeeea",
            selected_color_background="#97ecf8",
            text_color_active="white",
        )

        data = {
            "screen 1": {"text": "Mail", "icon": "gmail"},
            "screen 2": {"text": "Discord", "icon": "discord"},
            "screen 3": {"text": "LinkedIN", "icon": "linkedin"},
        }
        for key in data.keys():
            text = data[key]["text"]
            navigation_item = MDBottomNavigationItem(
                name=key, text=text, icon=data[key]["icon"]
            )
            navigation_item.add_widget(MDLabel(text=text, halign="center"))
            bottom_navigation.add_widget(navigation_item)

        screen.add_widget(bottom_navigation)
        return screen

Example().run()
```



Take a look at the above code example. This is a very simple UI. But looking at this code, you will not be able to figure the widget tree and understand which UI this code implements. This is named imperative programming style, which is used in Kivy.

Now let's see how the same code is implemented using the KV language, which uses a declarative style of describing widget properties.

Declarative style with KV language

```
from kivy.lang import Builder

from kivymd.app import MDApp

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

    MDBottomNavigation:
        panel_color: "#eeeeaea"
        selected_color_background: "#97ecf8"
        text_color_active: "white"

        MDBottomNavigationItem:
            name: "screen 1"
            text: "Mail"
            icon: "gmail"
            '''
        )
```

(continues on next page)

(continued from previous page)

```
        MDLabel:
            text: "Mail"
            halign: "center"

        MDBottomNavigationItem:
            name: "screen 2"
            text: "Discord"
            icon: "discord"

        MDLabel:
            text: "Discord"
            halign: "center"

        MDBottomNavigationItem:
            name: "screen 3"
            text: "LinkedIN"
            icon: "linkedin"

        MDLabel:
            text: "LinkedIN"
            halign: "center"
    ...
)
```

```
Test().run()
```

Mail



Looking at this code, we can now clearly see the widget tree and their properties. We can quickly navigate through the components of the screen and quickly change/add new properties/widgets. This is named declarative UI creation style.

But now the KivyMD library allows you to write Python code in a declarative style. Just as it is implemented in Flutter/Jetpack Compose/SwiftUI.

Declarative style with Python code

```
from kivymd.app import MDApp
from kivymd.uix.bottomnavigation import MDBottomNavigation, MDBottomNavigationItem
from kivymd.uix.label import MDLabel
from kivymd.uix.screen import MDScreen

class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                MDBottomNavigation(
                    MDBottomNavigationItem(
                        MDLabel(
                            text="Mail",
                            halign="center",
                        ),
                        name="screen 1",
                        text="Mail",
                        icon="gmail",
                    ),
                    MDBottomNavigationItem(
                        MDLabel(
                            text="Discord",
                            halign="center",
                        ),
                        name="screen 2",
                        text="Discord",
                        icon="discord",
                    ),
                    MDBottomNavigationItem(
                        MDLabel(
                            text="LinkedIN",
                            halign="center",
                        ),
                        name="screen 3",
                        text="LinkedIN",
                        icon="linkedin",
                    ),
                    panel_color="#eeeeea",
                    selected_color_background="#97ecf8",
                    text_color_active="white",
                )
            )
        )

Example().run()
```

Note: The KivyMD library does not support creating Kivy widgets in Python code in a declarative style.

But you can still use the declarative style of creating Kivy widgets in Python code. To do this, you need to create a new class that will be inherited from the Kivy widget and the *DeclarativeBehavior* class:

```
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button

from kivymd.app import MDApp
from kivymd.uix.behaviors import DeclarativeBehavior

class DeclarativeStyleBoxLayout(DeclarativeBehavior, BoxLayout):
    pass

class Example(MDApp):
    def build(self):
        return (
            DeclarativeStyleBoxLayout(
                Button(),
                Button(),
                orientation="vertical",
            )
        )

Example().run()
```

Get objects by identifiers

In the declarative style in Python code, the `ids` parameter of the specified widget will return only the id of the child widget/container, ignoring other ids. Therefore, to get objects by identifiers in declarative style in Python code, you must specify all the container ids in which the widget is nested until you get to the desired id:

```
from kivymd.app import MDApp
from kivy.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.floatlayout import MDFloatLayout

class Example(MDApp):
    def build(self):
        return (
            MDBoxLayout(
                MDFloatLayout(
                    MDRaisedButton(
                        id="button_1",
                        text="Button 1",
                        pos_hint={"center_x": 0.5, "center_y": 0.5},
                    ),
                ),
            )
```

(continues on next page)

(continued from previous page)

```

        id="box_container_1",
    ),
    MDBoxLayout(
        MDFloatLayout(
            MDRaisedButton(
                id="button_2",
                text="Button 2",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            ),
            id="float_container",
        ),
        id="box_container_2",
    )
)

def on_start(self):
    # {
    #     'box_container_1': <kivymd.uix.floatlayout.MDFloatLayout>,
    #     'box_container_2': <kivymd.uix.boxlayout.MDBoxLayout object>
    # }
    print(self.root.ids)

    # <kivymd.uix.button.button.MDRaisedButton>
    print(self.root.ids.box_container_2.ids.float_container.ids.button_2)

```

Example().run()

Yes, this is not a very good solution, but I think it will be fixed soon.

Warning: Declarative programming style in Python code in the KivyMD library is an experimental feature. Therefore, if you receive errors, do not hesitate to create new issue in the KivyMD repository.

API - `kivymd.uix.behaviors.declarative_bahavior`

class `kivymd.uix.behaviors.declarative_bahavior.DeclarativeBehavior(*args, **kwargs)`

Implements the creation and addition of child widgets as declarative programming style.

id

Widget ID.

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

2.5.7 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 kivy.lang import Builder  
  
from kivymd.app import MDApp  
  
KV = '''  
<MagicButton@MagicBehavior+MDRectangleFlatButton>  
  
MDFloatLayout:  
  
    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}
```

(continues on next page)

(continued from previous page)

```

    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)

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.

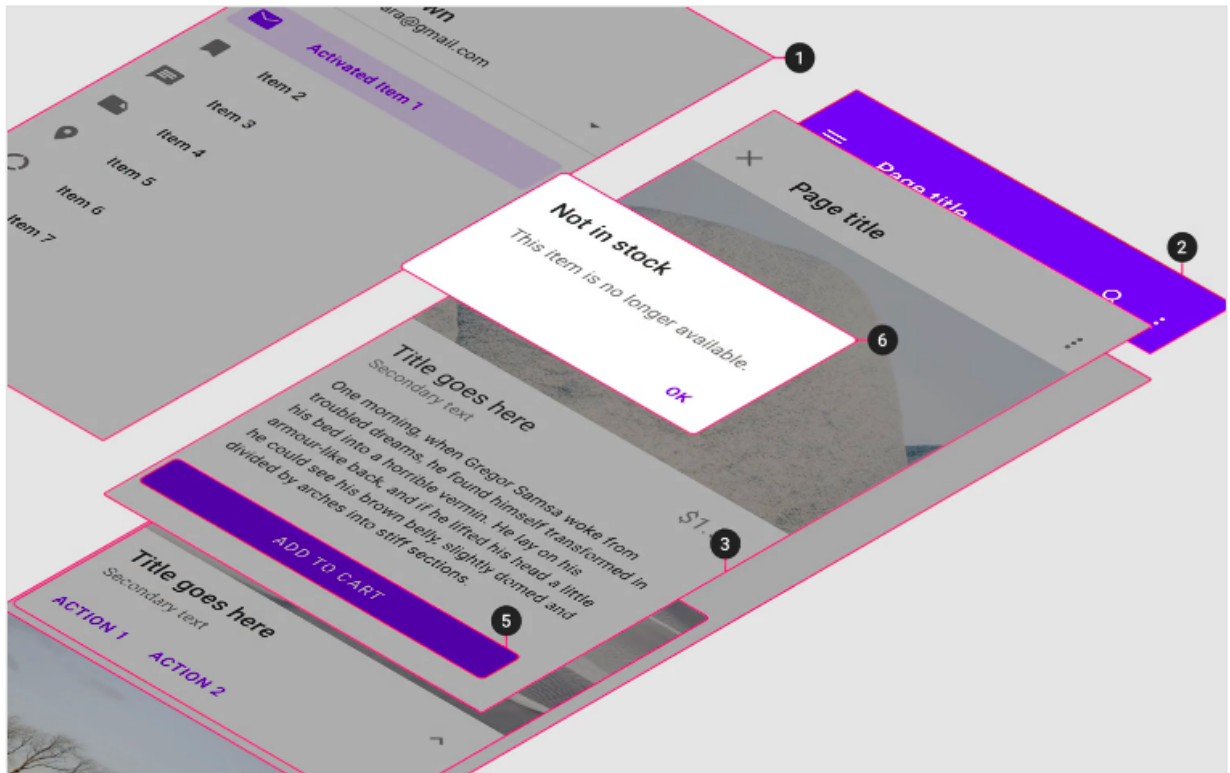
on_touch_up(self, *args)

2.5.8 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

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@MDBoxLayout>
    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:

    MDBoxLayout:
        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

```

(continues on next page)

(continued from previous page)

```

        NewShadowWidget:
            MDLabel:
                text: "It takes a lot of resources"
                adaptive_size: True
    '''

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 (

```

(continues on next page)

(continued from previous page)

```

    RectangularRippleBehavior,
    BackgroundColorBehavior,
    FakeRectangularElevationBehavior,
)

KV = '''
<RectangularElevationButton>:
    size_hint: None, None
    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 kivy.uix.behaviors import (
    CircularRippleBehavior,
    FakeCircularElevationBehavior,
)

```

(continues on next page)

(continued from previous page)

```

KV = '''
<CircularElevationButton>:
    size_hint: None, None
    size: "100dp", "100dp"
    radius: self.size[0] / 2
    md_bg_color: 0, 0, 1, 1

    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,
    MDBoxLayout,
):
    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 MDBoxLayout

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,
    MDBoxLayout,
):
    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)

```

(continues on next page)

(continued from previous page)

```

        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 MDBoxLayout
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, MDBoxLayout):
    pass

class Example(MDApp):

```

(continues on next page)

(continued from previous page)

```
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 MDAApp
from kivymd.uix.behaviors import CircularElevationBehavior, \
    CircularRippleBehavior
from kivymd.uix.boxlayout import MDBoxLayout

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:
```

(continues on next page)

(continued from previous page)

```

WidgetWithShadow:
    size: [root.size[1] / 2] * 2
    pos_hint: {"center": [0.5, 0.5]}
...

class WidgetWithShadow(
    CircularElevationBehavior,
    CircularRippleBehavior,
    ButtonBehavior,
    MDBoxLayout,
):
    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()

```

`elevation` is an `BoundedNumericProperty` and defaults to `0`.

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 canvas layer.

Incorrect:

```

<TiltedWidget>
    canvas.before:
        PushMatrix
        [...]
    canvas:
        PopMatrix

```

Correct:

```
<TiltedWidget>
    canvas.before:
        PushMatrix
        [...]
        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 (d'ont 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`r 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,
```

(continues on next page)

(continued from previous page)

```

    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.

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.

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

2.5.9 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 MDBoxLayout

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(MDBoxLayout, 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.focus_behavior.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 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.

2.6 Effects

2.6.1 StiffScrollEffect

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.stiffscroll

class kivymd.effects.stiffscroll.stiffscroll.StiffScrolleffect(**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.

```

2.6.2 FadingEdgeEffect

New in version 1.0.0.

The *FadingEdgeEffect* class implements a fade effect for *KivyMD* widgets:

```

from kivy.lang import Builder
from kivy.uix.scrollview import ScrollView

from kivymd.app import MDApp
from kivymd.effects.fadingedge.fadingedge import FadingEdgeEffect
from kivymd.uix.list import OneLineListItem

KV = '''
MDScreen:

    FadeScrollView:
        fade_height: self.height / 2
        fade_color: root.md_bg_color

        MDList:
            id: container
'''

class FadeScrollView(FadingEdgeEffect, ScrollView):
    pass

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()

```

Note: Use the same color value for the `fade_color` parameter as for the parent widget.

API - `kivymd.effects.fadingedge.fadingedge`

class `kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect(**kwargs)`

The class implements the fade effect.

New in version 1.0.0.

fade_color

Fade color.

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

fade_height

Fade height.

fade_height is an `ColorProperty` and defaults to `0`.

edge_top

Display fade edge top.

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

edge_bottom

Display fade edge bottom.

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

set_fade(*self*, *interval*: `Union[int, float]`)

Draws a bottom and top fade border on the canvas.

update_canvas(*self*, *instance_fadind_edge_effect*, *size*: `list[int, int]`, *rectangle_top*: `Rectangle`,
rectangle_bottom: `Rectangle`, *index*: `int`)

Updates the position and size of the fade border on the canvas. Called when the application screen is resized.

2.6.3 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.scrollview.ScrollView`:

```
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)
```

(continues on next page)

(continued from previous page)

```

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.roulettescroll`

class `kivymd.effects.roulettescroll.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 framerate and friction.

on_coasted_to_stop(*self*, **args*)

This event fires when the roulette has stopped, *making a selection*.

2.7 Templates

2.7.1 RotateWidget

New in version 1.0.0.

Base class for controlling the rotate of the widget.

Note: See [kivy.graphics.Rotate](#) for more information.

Kivy

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.app import App
from kivy.properties import NumericProperty
from kivy.uix.button import Button

KV = '''
Screen:

    RotateButton:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_rotate(self)

        canvas.before:
            PushMatrix
            Rotate:
                angle: self.rotate_value_angle
                axis: 0, 0, 1
                origin: self.center
        canvas.after:
            PopMatrix
'''

class RotateButton(Button):
    rotate_value_angle = NumericProperty(0)
```

(continues on next page)

(continued from previous page)

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

    def change_rotate(self, instance_button: Button) -> None:
        Animation(rotate_value_angle=45, d=0.3).start(instance_button)

Test().run()
```

KivyMD

```
from kivy.animation import Animation
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.templates import RotateWidget

KV = '''
MDScreen:

    RotateButton:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_rotate(self)
        elevation:0
'''

class RotateButton(MDRaisedButton, RotateWidget):
    pass

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

    def change_rotate(self, instance_button: MDRaisedButton) -> None:
        Animation(rotate_value_angle=45, d=0.3).start(instance_button)

Test().run()
```

API - `kivymd.uix.templates.rotatewidget.rotatewidget`**`class kivymd.uix.templates.rotatewidget.rotatewidget.RotateWidget`**

Base class for controlling the rotate of the widget.

`rotate_value_angle`

Property for getting/setting the angle of the rotation.

`rotate_value_angle` is an `NumericProperty` and defaults to `0`.

`rotate_value_axis`

Property for getting/setting the axis of the rotation.

`rotate_value_axis` is an `NumericProperty` and defaults to `(0, 0, 1)`.

2.7.2 ScaleWidget

New in version 1.0.0.

Base class for controlling the scale of the widget.

Note: See `kivy.graphics.Scale` for more information.

Kivy

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.uix.button import Button
from kivy.app import App

KV = '''
Screen:

    ScaleButton:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_scale(self)

        canvas.before:
            PushMatrix
            Scale:
                x: self.scale_value_x
                y: self.scale_value_y
                z: self.scale_value_x
                origin: self.center
        canvas.after:
            PopMatrix
'''
```

(continues on next page)

(continued from previous page)

```

class ScaleButton(Button):
    scale_value_x = NumericProperty(1)
    scale_value_y = NumericProperty(1)
    scale_value_z = NumericProperty(1)

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

    def change_scale(self, instance_button: Button) -> None:
        Animation(
            scale_value_x=0.5,
            scale_value_y=0.5,
            scale_value_z=0.5,
            d=0.3,
        ).start(instance_button)

Test().run()

```

KivyMD

```

from kivy.animation import Animation
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.templates import ScaleWidget

KV = '''
MDScreen:

    ScaleButton:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_scale(self)
        elevation:0
'''

class ScaleButton(MDRaisedButton, ScaleWidget):
    pass

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

```

(continues on next page)

(continued from previous page)

```
def change_scale(self, instance_button: MDRaisedButton) -> None:
    Animation(
        scale_value_x=0.5,
        scale_value_y=0.5,
        scale_value_z=0.5,
        d=0.3,
    ).start(instance_button)
```

```
Test().run()
```

API - kivymd.ui.templates.scalewidget.scalewidget

class kivymd.ui.templates.scalewidget.scalewidget.ScaleWidget

Base class for controlling the scale of the widget.

scale_value_x

X-axis value.

scale_value_x is an [NumericProperty](#) and defaults to 1.

scale_value_y

Y-axis value.

scale_value_y is an [NumericProperty](#) and defaults to 1.

scale_value_z

Z-axis value.

scale_value_z is an [NumericProperty](#) and defaults to 1.

2.7.3 StencilWidget

New in version 1.0.0.

Base class for controlling the stencil instructions of the widget.

Note: See [Stencil instructions](#) for more information.

Kivy

```
from kivy.lang import Builder
from kivy.app import App

KV = '''
Carousel:
```

(continues on next page)

(continued from previous page)

```

    Button:
        size_hint: .9, .8
        pos_hint: {"center_x": .5, "center_y": .5}

        canvas.before:
            StencilPush
            RoundedRectangle:
                pos: root.pos
                size: root.size
            StencilUse
        canvas.after:
            StencilUnUse
            RoundedRectangle:
                pos: root.pos
                size: root.size
            StencilPop
'''

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

Test().run()

```

KivyMD

```

from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.templates import StencilWidget
from kivymd.uix.fitimage import FitImage

KV = '''
MDCarousel:

    StencilImage:
        size_hint: .9, .8
        pos_hint: {"center_x": .5, "center_y": .5}
        source: "image.png"
'''

class StencilImage(FitImage, StencilWidget):
    pass

class Test(MDApp):
    def build(self):

```

(continues on next page)

(continued from previous page)

```
return Builder.load_string(KV)
```

```
Test().run()
```

API - `kivymd.uix.templates.stencilwidget.stencilwidget`

class `kivymd.uix.templates.stencilwidget.stencilwidget.StencilWidget`

Base class for controlling the stencil instructions of the widget

radius

Canvas radius.

New in version 1.0.0.

```
# Top left corner slice.
MDWidget:
    radius: [25, 0, 0, 0]
```

radius is an `VariableListProperty` and defaults to `[0, 0, 0, 0]`.

2.8 Changelog

2.8.1 1.0.2

See on GitHub: [tag 1.0.2 | compare 1.0.1/1.0.2](#)

```
pip install kivymd==1.0.2
```

- Bug fixes and other minor improvements.
- Added a button to copy the code to the documentation.
- Added the feature to view code examples of documentation in imperative and declarative styles.
- Added console scripts for developer tools.

2.8.2 1.0.1

See on GitHub: [tag 1.0.1 | compare 1.0.0/1.0.1](#)

```
pip install kivymd==1.0.1
```

- Bug fixes and other minor improvements.
- Fix <https://github.com/kivymd/KivyMD/issues/1305>.

2.8.3 1.0.0

See on GitHub: [tag 1.0.0](#) | [compare 0.104.2/1.0.0](#)

```
pip install kivymd==1.0.0
```

- 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;
- Added *animation_tooltip_dismiss* function and *on_dismiss* event to *MDTooltip* class;
- Added *MDColorPicker* component;
- Added new *transition* package - a set of classes for implementing transitions between application screens;
- Now all modules from the *uix* directory are packages;
- Type hints have been added to the source code of the KivyMD library;
- Added *divider_color* attribute to *BaseListItem* class;
- Added *load_all_kv_files* method to *MDDApp* class;
- Added *Templates* package - base classes for controlling the scale, rotation of the widget, etc.;
- Added *kivymd/tools/patterns* package - scripts for creating projects with design patterns;
- *FitImage* widget move from *kivymd.utils* to *kivymd.uix.fitimage*;
- Added *background_color_header*, *background_color_cell*, *background_color_selected_cell*, added methods for adding/removing rows to a common table to *MDDDataTable* widget;
- Added method for *update rows* to *MDDDataTable* class;
- Delete *kivymd/utils/hot_reload_viewer.py*;
- Added *kivymd/tools/hotreload* package;
- Added *top* value to *position* parameter of *MDDropdownMenu* class;
- Added *get_current_tab* method to *MDTabs* class;
- Added the feature to automatically create a virtual environment when creating a project using the *kivymd.tools.patterns.create_project* tool;
- Added the feature to use the *left_icon* for *MDTextField* text fields;
- The design and behavior of the *MDChip* widget is close to the material design spec;
- Added the feature to set the thickness of the *MDProgressBar* class;
- Added localization support when creating a project using the *create_project* tool;
- Added support *Material Design v3*;
- Added support badge icon to *MDIcon* class;
- Added the feature to use a radius for the *BaseListItem* class;
- *MDFloatingActionButton* class configured according to M3 style;
- Ripple animation for round buttons customized to material design standards;

- Fix *Warning, too much iteration done before the next frame* for button classes;
- Added `FadingEdgeEffect` class
- Added `MDSliverAppBar` widget;
- Added the feature to use `custom icons` and `font name` for the `MDBottomNavigation` class;
- Rename `MDToolbarr` to `MDTopAppBar` class;
- The `overflow behavior` from the `ActionBar` class of the *Kivy* framework has been added to the `MDTopAppBar` class;
- Add `shift_right` and `shift_left` attributes to `MDTooltip` class;
- Fixed the size of the `MDIconButton` icon when changing `icon_size` on mobile devices;
- Add new `MDSegmentedControl` widget;
- Add `on_release/on_press` events to `MDSmartTile` class;
- Add `mipmap` property to `FitImage` class;
- Added the feature to use `Hero` animation;
- Added `MDResponsiveLayout` layout;
- Added `add_view` utility;
- Added the feature to create widgets in `declarative programming style`;

2.8.4 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/ui/useranimationcard.py*
- Added usage types for *MDNavigationDrawer* 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 *MDRectangleFlatButton* class
- Add *MDRelativeLayout* class
- Added the feature to use radius for *MDNavigationDrawer* corners
- Removed *UserAnimationCard* class
- Added feature to set background color for *MDDialog* class
- Added *MDNavigationRail* 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 *MDDatePicker* 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.

2.8.5 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 *MDBoxLayout* classes
- Add *TouchBehavior* class
- Add *radius* parameter to *BackgroundColorBehavior* class
- Add *MDScreen* class
- Add *MDFloatLayout* class
- Added a *MDTextField* 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 *MDTextField* classes
- Added the feature to use ripple and focus behavior in *MDCard* class
- *MDDialogs* class redesigned to meet material design requirements
- Added *MDDDataTable* class

2.8.6 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.MDExpansionPanel` 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.MDToolbar` 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.bottomnavigation.MDBottomNavigation` class.
- Added feature to use panel type in the `kivymd.uix.expansionpanel.MDExpansionPanel` class: `kivymd.uix.expansionpanel.MDExpansionPanelOneLine`, `kivymd.uix.expansionpanel.MDExpansionPanelTwoLine` or `kivymd.uix.expansionpanel.MDExpansionPanelThreeLine`.

- Fixed panel opening animation in the `kivymd.uix.expansionpanel.MDExpansionPanel` 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

2.8.7 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.

2.8.8 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.

2.8.9 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.ui.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.

2.8.10 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 *ui* and *behaviors* folder to *package_data*.

2.8.11 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)).

2.8.12 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*.

2.8.13 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.

2.8.14 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.

2.8.15 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.

2.8.16 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.

2.8.17 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.

2.8.18 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/15c707dc4cfc2b6455f>).
- 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.

2.8.19 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.

2.8.20 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*.

2.8.21 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*.

2.8.22 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 *MDNavigationDrawer.use_logo*.

2.8.23 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*.

2.8.24 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 *MDFillRoundFlatIconButton* class.

2.8.25 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.

2.8.26 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>).

2.8.27 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).

2.8.28 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 *MDAccordionListItem* class.

2.8.29 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).

2.8.30 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 *MDTextFieldRound* class.

2.9 About

2.9.1 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

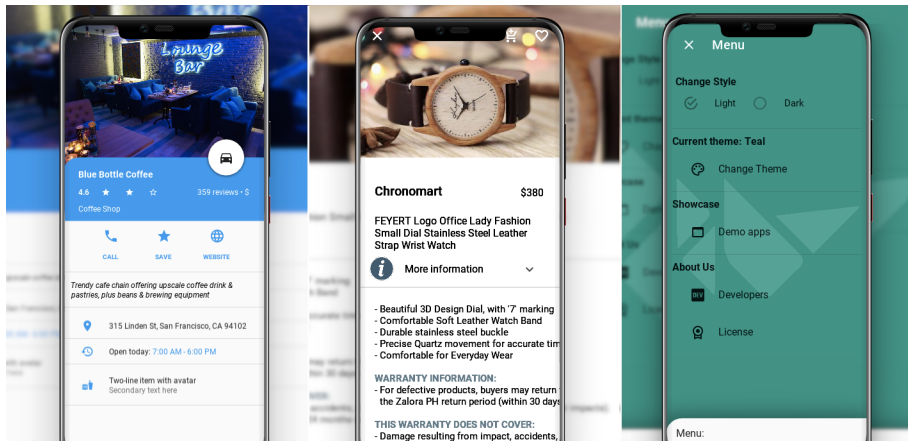
Copyright (c) 2020 Benedikt Zwölfer - fitimage module

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 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.

2.10 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.

This library is a fork of the [KivyMD project](#). We found the strength and brought this project to a new level.

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.

2.10.1 API - kivymd

`kivymd.release = True`

`kivymd.path`

Path to KivyMD package directory.

`kivymd.fonts_path`

Path to fonts directory.

`kivymd.images_path`

Path to images directory.

`kivymd.uix_path`

Path to uix directory.

2.10.2 Submodules

Register KivyMD widgets to use without import.

Register KivyMD widgets to use without import.

API - kivymd.factory_registers

kivymd.factory_registers.register

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/#relative-luminance> 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

Effects

API - `kivymd.effects`

Submodules

`kivymd.effects.fadingedge`

API - `kivymd.effects.fadingedge`

Submodules

`kivymd.effects.roulettесcroll`

API - `kivymd.effects.roulettесcroll`

Submodules

`kivymd.effects.stiffscroll`

API - `kivymd.effects.stiffscroll`

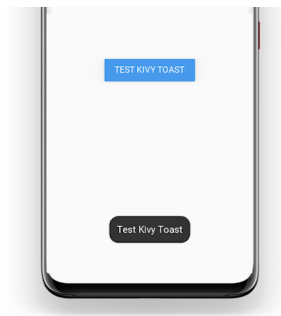
Submodules

`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 MDAApp

KV = '''
MDScreen:

    MDFlatButton:
        text: "My Toast"
        pos_hint:{"center_x": .5, "center_y": .5}
        on_press: app.show_toast()
'''

class Test(MDAApp):
    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.**toast**(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:

    MDTopAppBar:
        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):
```

(continues on next page)

(continued from previous page)

```

        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'.

Changed in version 2.1.0: Marked *attach_to* property as deprecated.

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_label: Label, texture_size: List[int])

Resizes the text if the text texture is larger than the screen size. Sets the size of the toast according to the texture size of the toast text.

toast(self, text_toast: str)

Displays a toast.

on_open(self)

Default open event handler.

fade_in(self)

Animation of opening toast on the screen.

fade_out(self, *args)

Animation of hiding toast on the screen.

on_touch_down(self, touch)

touch down event handler.

`kivymd.toast.kivytoast.kivytoast.toast(text: str = "", background: list = None, duration: float = 2.5) → None`

Displays a toast.

Parameters

- **text** – text to be displayed in the toast;
- **duration** – the amount of time (in seconds) that the toast is visible on the screen
- **background** – toast background color in `rgba` format;

kivymd.tools

API - kivymd.tools

Submodules

kivymd.tools.argument_parser

API - kivymd.tools.argument_parser

```
class kivymd.tools.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,
                                                         exit_on_error=True)
```

Object for parsing command line strings into Python objects.

Keyword Arguments:

- **prog** – The name of the program (default: `os.path.basename(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
- **`exit_on_error`** – Determines whether or not `ArgumentParser` exits with error info when an error occurs

`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.hotreload

API - `kivymd.tools.hotreload`

Submodules

HotReload

New in version 1.0.0.



Hot reload tool - is a fork of the project <https://github.com/tito/kaki>

Note: Since the project is not developing, we decided to include it in the KivvMD library and hope that the further development of the hot reload tool in the KivyMD project will develop faster.

This library enhance Kivy frameworks with opiniated features such as:

- Auto reloading kv or py (watchdog required, limited to some uses cases);
- Idle detection support;
- Foreground lock (Windows OS only);

Usage

Note: See [create project with hot reload](#) for more information.

TODO

- Add automatic reloading of Python classes;
- Add save application state on reloading;

FIXME

- On Windows, hot reloading of Python files may not work;

API - `kivymd.tools.hotreload.app`

`kivymd.tools.hotreload.app.original_argv`

`kivymd.tools.hotreload.app.monotonic`

`kivymd.tools.hotreload.app.PY3 = True`

class `kivymd.tools.hotreload.app.ExceptionClass`

Base handler that catches exceptions in `runTouchApp()`. You can subclass and extend it as follows:

```
class E(ExceptionHandler):
    def handle_exception(self, inst):
        Logger.exception('Exception caught by ExceptionHandler')
        return ExceptionManager.PASS

ExceptionManager.add_handler(E())
```

Then, all exceptions will be set to PASS, and logged to the console!

handle_exception(*self*, *inst*)

Called by `ExceptionManagerBase` to handle a exception.

Defaults to returning `ExceptionManager.RAISE` that re-raises the exception. Return `ExceptionManager.PASS` to indicate that the exception was handled and should be ignored.

This may be called multiple times with the same exception, if `ExceptionManager.RAISE` is returned as the exception bubbles through multiple kivy exception handling levels.

class kivymd.tools.hotreload.app.**MDApp**(**kwargs)

HotReload Application class.

DEBUG

Control either we activate debugging in the app or not. Defaults depend if ‘DEBUG’ exists in os.environ.

DEBUG is a *BooleanProperty*.

FOREGROUND_LOCK

If *True* it will require the foreground lock on windows.

FOREGROUND_LOCK is a *BooleanProperty* and defaults to *False*.

KV_FILES

List of KV files under management for auto reloader.

KV_FILES is a *ListProperty* and defaults to *[]*.

KV_DIRS

List of managed KV directories for autoloader.

KV_DIRS is a *ListProperty* and defaults to *[]*.

AUTORELOADER_PATHS

List of path to watch for auto reloading.

AUTORELOADER_PATHS is a *ListProperty* and defaults to *([“.”, {“recursive”: True}])*.

AUTORELOADER_IGNORE_PATTERNS

List of extensions to ignore.

AUTORELOADER_IGNORE_PATTERNS is a *ListProperty* and defaults to *[‘*.pyc’, ‘__pycache__’]*.

CLASSES

Factory classes managed by hotreload.

CLASSES is a *DictProperty* and defaults to *{}*.

IDLE_DETECTION

Idle detection (if *True*, event on_idle/on_wakeup will be fired). Rearming idle can also be done with *rearm_idle()*.

IDLE_DETECTION is a *BooleanProperty* and defaults to *False*.

IDLE_TIMEOUT

Default idle timeout.

IDLE_TIMEOUT is a *NumericProperty* and defaults to *60*.

RAISE_ERROR

Raise error. When the *DEBUG* is activated, it will raise any error instead of showing it on the screen. If you still want to show the error when not in *DEBUG*, put this to *False*.

RAISE_ERROR is a *BooleanProperty* and defaults to *True*.

build(self)

Initializes the application; it will be called only once. If this method returns a widget (tree), it will be used as the root widget and added to the window.

Returns

None or a root *Widget* instance if no self.root exists.

get_root(self)

Return a root widget, that will contains your application. It should not be your application widget itself, as it may be destroyed and recreated from scratch when reloading.

By default, it returns a RelativeLayout, but it could be a Viewport.

get_root_path(self)

Return the root file path.

abstract build_app(self, first=False)

Must return your application widget.

If *first* is set, it means that will be your first time ever that the application is built. Act according to it.

unload_app_dependencies(self)

Called when all the application dependencies must be unloaded. Usually happen before a reload

load_app_dependencies(self)

Load all the application dependencies. This is called before rebuild.

rebuild(self, *args, **kwargs)**set_error(self, exc, tb=None)****bind_key(self, key, callback)**

Bind a key (keycode) to a callback (cannot be unbind).

property appname(self)

Return the name of the application class.

enable_autoreload(self)

Enable autoreload manually. It is activated automatically if “DEBUG” exists in environ. It requires the *watchdog* module.

prepare_foreground_lock(self)

Try forcing app to front permanently to avoid windows pop ups and notifications etc.app.

Requires fake full screen and borderless.

Note: This function is called automatically if *FOREGROUND_LOCK* is set

set_widget(self, wid)

Clear the root container, and set the new approot widget to *wid*.

apply_state(self, state)

Whatever the current state is, reapply the current state.

install_idle(self, timeout=60)

Install the idle detector. Default timeout is 60s. Once installed, it will check every second if the idle timer expired. The timer can be rearm using [rearm_idle\(\)](#).

rearm_idle(self, *args)

Rearm the idle timer.

patch_builder(self)**on_idle(self, *args)**

Event fired when the application enter the idle mode.

`on_wakeup(self, *args)`

Event fired when the application leaves idle mode.

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",
```

(continues on next page)

(continued from previous page)

```

    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, images and KV files 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.patterns`

API - `kivymd.tools.patterns`

Submodules

The script creates a new View package

The script creates a new View package in an existing project with an MVC template created using the `create_project` utility.

New in version 1.0.0.

See also:

Utility `create_project`

Use a clean architecture for your applications.

To add a new view to an existing project that was created using the *create_project* utility, use the following command:

```
python -m kivymd.tools.patterns.add_view \  
    name_pattern \  
    path_to_project \  
    name_view
```

Example command:

```
python -m kivymd.tools.patterns.add_view \  
    MVC \  
    /Users/macbookair/Projects \  
    NewScreen
```

You can also add new views with responsive behavior to an existing project:

```
python -m kivymd.tools.patterns.add_view \  
    MVC \  
    /Users/macbookair/Projects \  
    NewScreen \  
    --use_responsive yes
```

For more information about adaptive design, [see here](#).

API - `kivymd.tools.patterns.add_view`

`kivymd.tools.patterns.add_view.main()`

The function of adding a new view to the project.

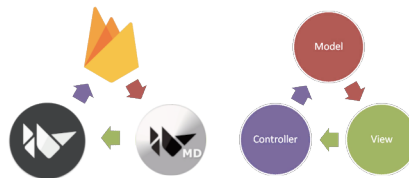
Script creates a project with the MVC pattern

New in version 1.0.0.

See also:

[MVC pattern](#)

Use a clean architecture for your applications.



Use a clean architecture for your applications. KivyMD allows you to quickly create a project template with the MVC pattern. So far, this is the only pattern that this utility offers. You can also include database support in your project. At the moment, support for the Firebase database (the basic implementation of the real time database) and RestDB (the full implementation) is available.

Project creation

Template command:

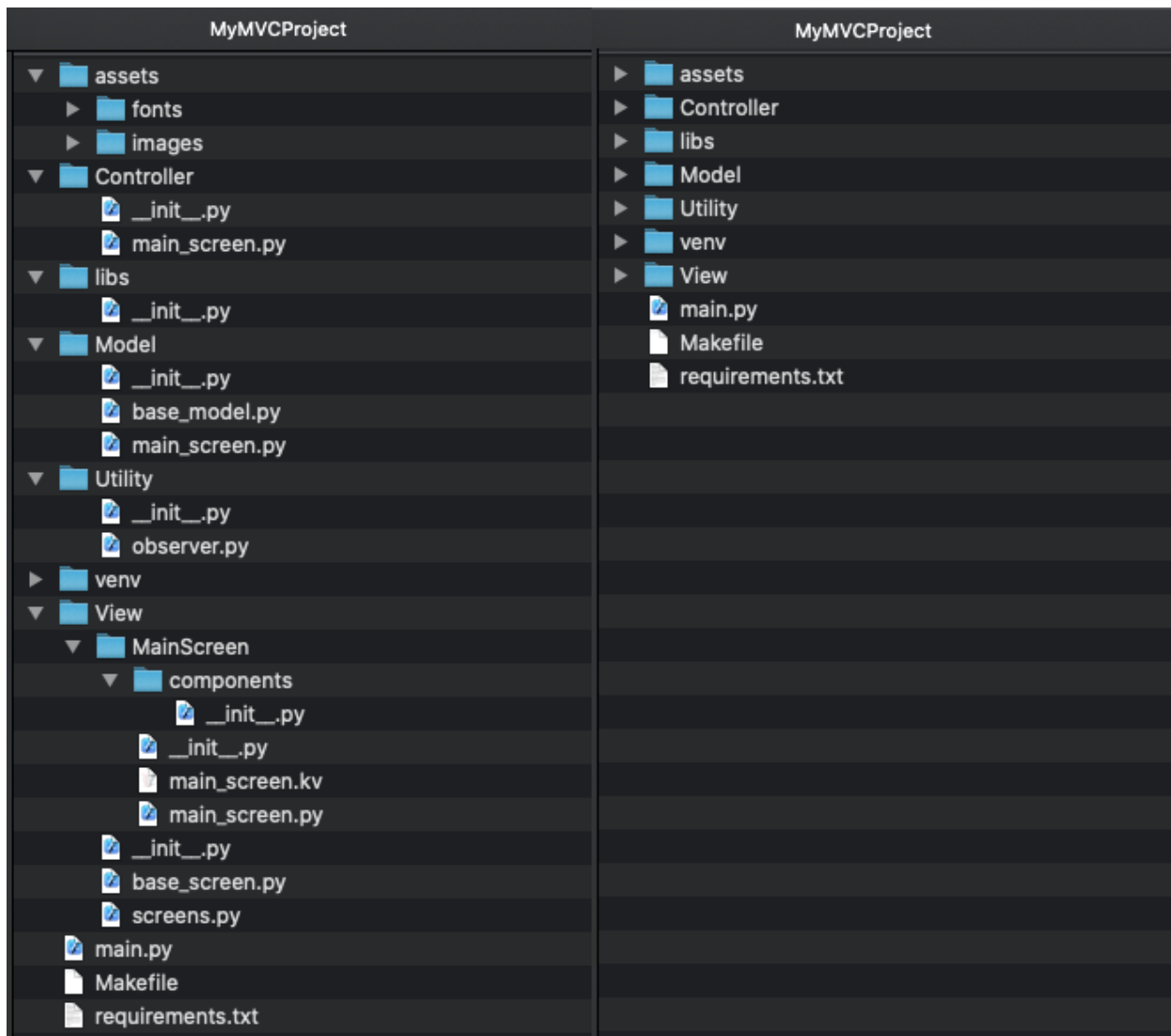
```
kivymd.create_project \  
    name_pattern \  
    path_to_project \  
    name_project \  
    python_version \  
    kivy_version
```

Example command:

```
kivymd.create_project \  
    MVC \  
    /Users/macbookair/Projects \  
    MyMVCProject \  
    python3.10 \  
    2.1.0
```

This command will by default create a project with an MVC pattern. Also, the project will create a virtual environment with Python 3.10, Kivy version 2.1.0 and KivyMD master version.

Note: Please note that the Python version you specified must be installed on your computer.



Creating a project using a database

Note: Note that in the following command, you can use one of two database names: ‘firebase’ or ‘restdb’.

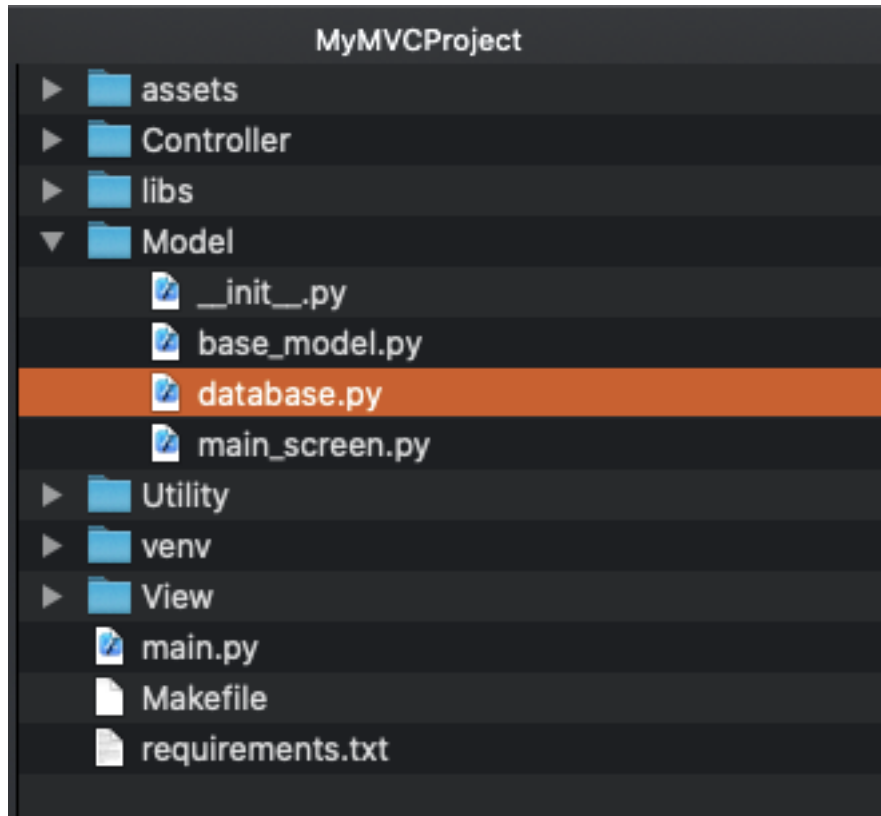
Template command:

```
kivymd.create_project \
  name_pattern \
  path_to_project \
  name_project \
  python_version \
  kivy_version \
  --name_database
```

Example command:

```
kivymd.create_project \
    MVC \
    /Users/macbookair/Projects \
    MyMVCProject \
    python3.10 \
    2.1.0 \
    --name_database restdb
```

This command will create a project with an MVC template by default. The project will also create a virtual environment with Python 3.10, Kivy version 2.1.0, KivyMD master version and a wrapper for working with the database restdb.io.



```
class DataBase:
    def __init__(self):
        database_url = "https://restdbio-5498.restdb.io"
        api_key = "7ce258d66f919d3a891d1166558765f0b4dbd"
```

Note: Please note that *database.py* the shell in the *DataBase* class uses the *database_url* and *api_key* parameters on the test database (works only in read mode), so you should use your data for the database.

Create project with hot reload

Template command:

```
kivymd.create_project \  
    name_pattern \  
    path_to_project \  
    name_project \  
    python_version \  
    kivy_version \  
    --use_hotreload
```

Example command:

```
kivymd.create_project \  
    MVC \  
    /Users/macbookair/Projects \  
    MyMVCProject \  
    python3.10 \  
    2.1.0 \  
    --use_hotreload yes
```

After creating the project, open the file *main.py*, there is a lot of useful information. Also, the necessary information is in other modules of the project in the form of comments. So do not forget to look at the source files of the created project.

Create project with responsive view

When creating a project, you can specify which views should use responsive behavior. To do this, specify the name of the view/views in the *--use_responsive* argument:

Template command:

```
kivymd.create_project \  
    name_pattern \  
    path_to_project \  
    name_project \  
    python_version \  
    kivy_version \  
    --name_screen FirstScreen SecondScreen ThirdScreen \  
    --use_responsive FirstScreen SecondScreen
```

The *FirstScreen* and *SecondScreen* views will be created with an responsive architecture. For more detailed information about using the adaptive view, see the [MDResponsiveLayout](#) widget.

Others command line arguments

Required Arguments

- **pattern**
 - the name of the pattern with which the project will be created
- **directory**
 - directory in which the project will be created
- **name**
 - project name
- **python_version**
 - the version of Python (specify as *python3.9* or *python3.8*) with
 - which the virtual environment will be created
- **kivy_version**
 - version of Kivy (specify as *2.1.0* or *master*) that will be used in the project

Optional arguments

- **name_screen**
 - the name of the class which be used when creating the project pattern

When you need to create an application template with multiple screens, use multiple values separated by a space for the *name_screen* parameter, for example, as shown below:

Template command:

```
kivymd.create_project \
  name_pattern \
  path_to_project \
  name_project \
  python_version \
  kivy_version \
  --name_screen FirstScreen SecondScreen ThirdScreen
```

- **name_database**
 - provides a basic template for working with the ‘firebase’ library
 - or a complete implementation for working with a database ‘restdb.io’
- **use_hotreload**
 - creates a hot reload entry point to the application
- **use_localization**
 - creates application localization files
- **use_responsive**
 - the name/names of the views to be used by the responsive UI

Warning: On Windows, hot reloading of Python files may not work. But, for example, there is no such problem in macOS. If you fix this, please report it to the KivyMD community.

API - `kivymd.tools.patterns.create_project`

`kivymd.tools.patterns.create_project.main()`

Project creation function.

`kivymd.tools.patterns.MVC`

API - `kivymd.tools.patterns.MVC`

Submodules

`kivymd.tools.patterns.MVC.Model`

API - `kivymd.tools.patterns.MVC.Model`

Submodules

`kivymd.tools.patterns.MVC.Model.database_firebase`

API - `kivymd.tools.patterns.MVC.Model.database_firebase`

`kivymd.tools.patterns.MVC.Model.database_firebase.get_connect`(*func*, *host*='8.8.8.8', *port*=53, *timeout*=3)

Checks for an active Internet connection.

class `kivymd.tools.patterns.MVC.Model.database_firebase.DataBase`

Your methods for working with the database should be implemented in this class.

name = `Firestore`

get_data_from_collection(*self*, *name_collection*: *str*)

Returns data of the selected collection from the database.

Restdb.io API Wrapper

This package is an API Wrapper for the website restdb.io, which allows for online databases.

API - kivymd.tools.patterns.MVC.Model.database_restdb

`kivymd.tools.patterns.MVC.Model.database_restdb.get_connect(func, host='8.8.8.8', port=53, timeout=3)`

Checks for an active Internet connection.

class `kivymd.tools.patterns.MVC.Model.database_restdb.DataBase`

name = RestDB

upload_file(*self*, *path_to_file*: *str*)

Uploads a file to the database. You can upload a file to the database only from a paid account.

get_data_from_collection(*self*, *collection_address*: *str*)

Returns data of the selected collection from the database.

delete_doc_from_collection(*self*, *collection_address*: *str*)

Delete data of the selected collection from the database.

Parameters

collection_address – “database_url/id_collection”.

add_doc_to_collection(*self*, *data*: *dict*, *collection_address*: *str*)

Add collection to the database.

edit_data(*self*, *collection*: *dict*, *collection_address*: *str*, *collection_id*: *str*)

Modifies data in a collection of data in a database.

kivymd.tools.patterns.MVC.libs**API - kivymd.tools.patterns.MVC.libs****Submodules****kivymd.tools.patterns.MVC.libs.translation****API - kivymd.tools.patterns.MVC.libs.translation**

class `kivymd.tools.patterns.MVC.libs.translation.Translation(defaultlang, domian, resource_dir)`

Original source - <https://github.com/tito/kivy-gettext-example>.

observers = []

fbind(*self*, *name*, *func*, *args*, ***kwargs*)

funbind(*self*, *name*, *func*, *args*, ***kwargs*)

switch_lang(*self*, *lang*)

kivymd.tools.release

API - kivymd.tools.release

Submodules

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, md_widget, value: bool)

`on_adaptive_width(self, md_widget, value: bool)`

`on_adaptive_size(self, md_widget, value: bool)`

Submodules

kivymd.uix.backdrop

API - kivymd.uix.backdrop

Submodules

kivymd.uix.banner

API - kivymd.uix.banner

Submodules

Behaviors

Modules and classes implementing various behaviors for buttons etc.

API - kivymd.uix.behaviors

Submodules

kivymd.uix.bottomnavigation

API - kivymd.uix.bottomnavigation

Submodules

kivymd.uix.bottomsheet

API - kivymd.uix.bottomsheet

Submodules

kivymd.uix.button

API - kivymd.uix.button

Submodules

kivymd.uix.card

API - kivymd.uix.card

Submodules

kivymd.uix.chip

API - kivymd.uix.chip

Submodules

Controllers

New in version 1.0.0.

Modules and classes that implement useful methods for getting information about the state of the current application window.

API - kivymd.uix.controllers

Submodules

kivymd.uix.datatables

API - kivymd.uix.datatables

Submodules

kivymd.uix.dialog

API - kivymd.uix.dialog

Submodules

kivymd.uix.dropdownitem

API - kivymd.uix.dropdownitem

Submodules

kivymd.uix.expansionpanel

API - kivymd.uix.expansionpanel

Submodules

kivymd.uix.filemanager

API - [kivymd.uix.filemanager](#)

Submodules

kivymd.uix.fitimage

API - [kivymd.uix.fitimage](#)

Submodules

kivymd.uix.imagelist

API - [kivymd.uix.imagelist](#)

Submodules

kivymd.uix.label

API - [kivymd.uix.label](#)

Submodules

kivymd.uix.list

API - [kivymd.uix.list](#)

Submodules

kivymd.uix.menu

API - [kivymd.uix.menu](#)

Submodules

kivymd.uix.navigationdrawer

API - [kivymd.uix.navigationdrawer](#)

Submodules

kivymd.uix.navigationrail

API - `kivymd.uix.navigationrail`

Submodules

`kivymd.uix.pickers`

API - `kivymd.uix.pickers`

Submodules

`kivymd.uix.pickers.colorpicker`

API - `kivymd.uix.pickers.colorpicker`

Submodules

`kivymd.uix.pickers.datepicker`

API - `kivymd.uix.pickers.datepicker`

Submodules

`kivymd.uix.pickers.timepicker`

API - `kivymd.uix.pickers.timepicker`

Submodules

`kivymd.uix.progressbar`

API - `kivymd.uix.progressbar`

Submodules

`kivymd.uix.refreshlayout`

API - `kivymd.uix.refreshlayout`

Submodules

`kivymd.uix.segmentedcontrol`

API - `kivymd.uix.segmentedcontrol`

Submodules

kivymd.uix.selection

API - kivymd.uix.selection

Submodules

kivymd.uix.selectioncontrol

API - kivymd.uix.selectioncontrol

Submodules

kivymd.uix.slider

API - kivymd.uix.slider

Submodules

kivymd.uix.sliverappbar

API - kivymd.uix.sliverappbar

Submodules

kivymd.uix.snackbar

API - kivymd.uix.snackbar

Submodules

kivymd.uix.spinner

API - kivymd.uix.spinner

Submodules

kivymd.uix.swiper

API - kivymd.uix.swiper

Submodules

kivymd.uix.tab

API - kivymd.uix.tab

Submodules

Templates

Base classes for controlling the scale, rotation of the widget, etc.

API - `kivymd.uix.templates`

Submodules

`kivymd.uix.templates.rotatewidget`

API - `kivymd.uix.templates.rotatewidget`

Submodules

`kivymd.uix.templates.scalewidget`

API - `kivymd.uix.templates.scalewidget`

Submodules

`kivymd.uix.templates.stencilwidget`

API - `kivymd.uix.templates.stencilwidget`

Submodules

`kivymd.uix.textfield`

API - `kivymd.uix.textfield`

Submodules

`kivymd.uix.toolbar`

API - `kivymd.uix.toolbar`

Submodules

`kivymd.uix.tooltip`

API - `kivymd.uix.tooltip`

Submodules

kivymd.uix.transition

API - kivymd.uix.transition

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*)

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)

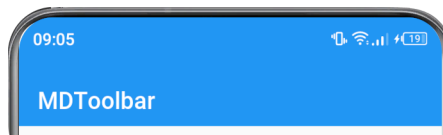
update_fps(self, *args)

kivymd.utils.setBarsColors**API - kivymd.utils.setBarsColors**

kivymd.utils.setBarsColors.**setBarsColors**(status_bar_color: Union[None, list],
navigation_bar_color: Union[None, list], icons_color: str
= 'Light')

Sets the color of the status of the StatusBar and NavigationBar.

Warning: Works only on Android devices.



```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.utils.setBarsColors import setBarsColors

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

    MDTopAppBar:
        title: "MDTopAppBar"

    MDBottomNavigation:
        panel_color: app.theme_cls.primary_color
        text_color_active: .2, .2, .2, 1
        text_color_normal: .9, .9, .9, 1
        use_text: False
```

(continues on next page)

(continued from previous page)

```

MDBottomNavigationItem:
    icon: 'gmail'

MDBottomNavigationItem:
    icon: 'twitter'

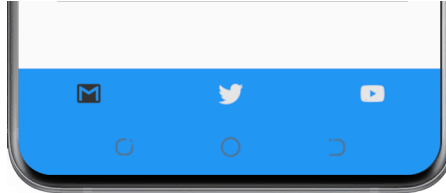
MDBottomNavigationItem:
    icon: 'youtube'
'''

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

    def setBarsColors(self):
        setBarsColors(
            self.theme_cls.primary_color, # status bar color
            self.theme_cls.primary_color, # navigation bar color
            "Light",                       # icons color of status bar
        )

Test().run()

```

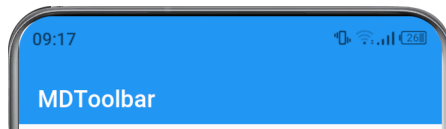


Dark icon mode

```

def setBarsColors(self):
    setBarsColors(
        self.theme_cls.primary_color, # status bar color
        self.theme_cls.primary_color, # navigation bar color
        "Dark",                       # icons color of status bar
    )

```



New in version 1.0.0.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

k

- kivy, 477
- kivy.app, 25
- kivy.color_definitions, 27
- kivy.effects, 478
- kivy.effects.fadingedge, 479
- kivy.effects.fadingedge.fadingedge, 455
- kivy.effects.roulettescroll, 479
- kivy.effects.roulettescroll.roulettescroll, 456
- kivy.effects.stiffscroll, 479
- kivy.effects.stiffscroll.stiffscroll, 453
- kivy.factory_registers, 477
- kivy.font_definitions, 32
- kivy.icon_definitions, 30
- kivy.material_resources, 478
- kivy.theming, 7
- kivy.theming_dynamic_text, 478
- kivy.toast, 479
- kivy.toast.androidtoast, 479
- kivy.toast.androidtoast.androidtoast, 480
- kivy.toast.kivytoast, 481
- kivy.toast.kivytoast.kivytoast, 481
- kivy.tools, 483
- kivy.tools.argument_parser, 483
- kivy.tools.hotreload, 484
- kivy.tools.hotreload.app, 484
- kivy.tools.packaging, 488
- kivy.tools.packaging.pyinstaller, 488
- kivy.tools.packaging.pyinstaller.hook-kivy, 489
- kivy.tools.patterns, 489
- kivy.tools.patterns.add_view, 489
- kivy.tools.patterns.create_project, 490
- kivy.tools.patterns.MVC, 496
- kivy.tools.patterns.MVC.libs, 497
- kivy.tools.patterns.MVC.libs.translation, 497
- kivy.tools.patterns.MVC.Model, 496
- kivy.tools.patterns.MVC.Model.database_firestore, 496
- kivy.tools.patterns.MVC.Model.database_restdb, 496
- kivy.tools.release, 498
- kivy.tools.release.git_commands, 498
- kivy.tools.release.make_release, 498
- kivy.tools.release.update_icons, 499
- kivy.uix, 500
- kivy.uix.anchorlayout, 33
- kivy.uix.backdrop, 501
- kivy.uix.backdrop.backdrop, 134
- kivy.uix.banner, 501
- kivy.uix.banner.banner, 192
- kivy.uix.behaviors, 501
- kivy.uix.behaviors.backgroundcolor_behavior, 423
- kivy.uix.behaviors.declarative_behavior, 429
- kivy.uix.behaviors.elevation, 438
- kivy.uix.behaviors.focus_behavior, 451
- kivy.uix.behaviors.hover_behavior, 421
- kivy.uix.behaviors.magic_behavior, 436
- kivy.uix.behaviors.ripple_behavior, 425
- kivy.uix.behaviors.toggle_behavior, 419
- kivy.uix.behaviors.touch_behavior, 417
- kivy.uix.bottomnavigation, 501
- kivy.uix.bottomnavigation.bottomnavigation, 381
- kivy.uix.bottomsheet, 501
- kivy.uix.bottomsheet.bottomsheet, 117
- kivy.uix.boxlayout, 57
- kivy.uix.button, 501
- kivy.uix.button.button, 100
- kivy.uix.card, 502
- kivy.uix.card.card, 197
- kivy.uix.carousel, 72
- kivy.uix.chip, 502
- kivy.uix.chip.chip, 409
- kivy.uix.circularlayout, 53
- kivy.uix.controllers, 502
- kivy.uix.controllers.windowcontroller, 416
- kivy.uix.datatables, 502
- kivy.uix.datatables.datatables, 79
- kivy.uix.dialog, 502

kivymd.uix.dialog.dialog, 362
kivymd.uix.dropdownitem, 502
kivymd.uix.dropdownitem.dropdownitem, 378
kivymd.uix.expansionpanel, 502
kivymd.uix.expansionpanel.expansionpanel, 248
kivymd.uix.filemanager, 503
kivymd.uix.filemanager.filemanager, 374
kivymd.uix.fitimage, 503
kivymd.uix.fitimage.fitimage, 310
kivymd.uix.floatlayout, 74
kivymd.uix.gridlayout, 71
kivymd.uix.hero, 62
kivymd.uix.imagelist, 503
kivymd.uix.imagelist.imagelist, 75
kivymd.uix.label, 503
kivymd.uix.label.label, 252
kivymd.uix.list, 503
kivymd.uix.list.list, 395
kivymd.uix.menu, 503
kivymd.uix.menu.menu, 258
kivymd.uix.navigationdrawer, 503
kivymd.uix.navigationdrawer.navigationdrawer, 313
kivymd.uix.navigationrail, 503
kivymd.uix.navigationrail.navigationrail, 155
kivymd.uix.pickers, 504
kivymd.uix.pickers.colorpicker, 504
kivymd.uix.pickers.colorpicker.colorpicker, 237
kivymd.uix.pickers.datepicker, 504
kivymd.uix.pickers.datepicker.datepicker, 217
kivymd.uix.pickers.timepicker, 504
kivymd.uix.pickers.timepicker.timepicker, 241
kivymd.uix.progressbar, 504
kivymd.uix.progressbar.progressbar, 184
kivymd.uix.recyclegridlayout, 35
kivymd.uix.recycleview, 59
kivymd.uix.refreshlayout, 504
kivymd.uix.refreshlayout.refreshlayout, 77
kivymd.uix.relativelayout, 61
kivymd.uix.responsivelayout, 51
kivymd.uix.screen, 56
kivymd.uix.screenmanager, 57
kivymd.uix.scrollview, 50
kivymd.uix.segmentedcontrol, 504
kivymd.uix.segmentedcontrol.segmentedcontrol, 357
kivymd.uix.selection, 505
kivymd.uix.selection.selection, 210
kivymd.uix.selectioncontrol, 505
kivymd.uix.selectioncontrol.selectioncontrol, 346
kivymd.uix.slider, 505
kivymd.uix.slider.slider, 96
kivymd.uix.sliverappbar, 505
kivymd.uix.sliverappbar.sliverappbar, 126
kivymd.uix.snackbar, 505
kivymd.uix.snackbar.snackbar, 303
kivymd.uix.spinner, 505
kivymd.uix.spinner.spinner, 278
kivymd.uix.stacklayout, 59
kivymd.uix.swiper, 505
kivymd.uix.swiper.swiper, 389
kivymd.uix.tab, 505
kivymd.uix.tab.tab, 281
kivymd.uix.taptargetview, 37
kivymd.uix.templates, 506
kivymd.uix.templates.rotatewidget, 506
kivymd.uix.templates.rotatewidget.rotatewidget, 459
kivymd.uix.templates.scalewidget, 506
kivymd.uix.templates.scalewidget.scalewidget, 461
kivymd.uix.templates.stencilwidget, 506
kivymd.uix.templates.stencilwidget.stencilwidget, 463
kivymd.uix.textfield, 506
kivymd.uix.textfield.textfield, 332
kivymd.uix.toolbar, 506
kivymd.uix.toolbar.toolbar, 139
kivymd.uix.tooltip, 506
kivymd.uix.tooltip.tooltip, 189
kivymd.uix.transition, 507
kivymd.uix.transition.transition, 380
kivymd.uix.widget, 34
kivymd.utils, 507
kivymd.utils.asynckivy, 507
kivymd.utils.fpsmonitor, 507
kivymd.utils.setBarsColors, 508

INDEX

A

`accent_color` (*kivymd.theming.ThemeManager* attribute), 15

`accent_color` (*kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker* attribute), 224

`accent_dark` (*kivymd.theming.ThemeManager* attribute), 16

`accent_dark_hue` (*kivymd.theming.ThemeManager* attribute), 15

`accent_hue` (*kivymd.theming.ThemeManager* attribute), 15

`accent_light` (*kivymd.theming.ThemeManager* attribute), 16

`accent_light_hue` (*kivymd.theming.ThemeManager* attribute), 15

`accent_palette` (*kivymd.theming.ThemeManager* attribute), 15

`active` (*kivymd.uix.chip.chip.MDChip* attribute), 416

`active` (*kivymd.uix.navigationrail.navigationrail.MDNavigationRailItem* attribute), 169

`active` (*kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox* attribute), 349

`active` (*kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch* attribute), 352

`active` (*kivymd.uix.slider.slider.MDSlider* attribute), 96

`active` (*kivymd.uix.spinner.spinner.MDSpinner* attribute), 281

`active_line` (*kivymd.uix.textfield.textfield.MDTextField* attribute), 339

`adaptive_height` (*kivymd.uix.MDAaptiveWidget* attribute), 500

`adaptive_size` (*kivymd.uix.MDAaptiveWidget* attribute), 500

`adaptive_width` (*kivymd.uix.MDAaptiveWidget* attribute), 500

`add_action_button_to_overflow()`
(*kivymd.uix.toolbar.toolbar.MDTopAppBar* method), 153

`add_actions_buttons()`
(*kivymd.uix.banner.banner.MDBanner* method), 196

`add_doc_to_collection()`
(*kivymd.tools.patterns.MVC.Model.database_restdb.DataBase* method), 497

`add_item()` (*kivymd.uix.bottomsheet.bottomsheet.MDGridBottomSheet* method), 126

`add_item()` (*kivymd.uix.bottomsheet.bottomsheet.MDListBottomSheet* method), 125

`add_overflow_button()`
(*kivymd.uix.toolbar.toolbar.MDTopAppBar* method), 153

`add_row()` (*kivymd.uix.datatables.datatables.MDDataTable* method), 92

`add_scrim()` (*kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer* method), 320

`add_widget()` (*kivymd.uix.backdrop.backdrop.MDBackdrop* method), 138

`add_widget()` (*kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar* method), 388

`add_widget()` (*kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet* method), 123

`add_widget()` (*kivymd.uix.card.card.MDCardSwipe* method), 209

`add_widget()` (*kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel* method), 252

`add_widget()` (*kivymd.uix.list.list.BaseListItem* method), 407

`add_widget()` (*kivymd.uix.list.list.MDList* method), 404

`add_widget()` (*kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer* method), 327

`add_widget()` (*kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer* method), 320

`add_widget()` (*kivymd.uix.navigationrail.navigationrail.MDNavigationRail* method), 184

`add_widget()` (*kivymd.uix.screenmanager.MDScreenManager* method), 57

`add_widget()` (*kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl* method), 361

`add_widget()` (*kivymd.uix.selection.selection.MDSelectionList* method), 216

`add_widget()` (*kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar* method), 133

`add_widget()` (*kivymd.uix.swiper.swiper.MDSwiper* method), 393

add_widget() (*kivymd.uix.tab.tab.MDTabs* method), 301
add_widget() (*kivymd.uix.toolbar.toolbar.MDBottomAppBar* method), 155
adjacent_color_constants (*kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker* attribute), 240
adjust_position() (*kivymd.uix.menu.menu.MDDropdownMenu* method), 278
adjust_tooltip_position() (*kivymd.uix.tooltip.tooltip.MDTooltip* method), 191
ajust_radius() (*kivymd.uix.menu.menu.MDDropdownMenu* method), 277
allow_stretch (*kivymd.uix.tab.tab.MDTabs* attribute), 300
am_pm (*kivymd.uix.pickers.timepicker.timepicker.MDTimePicker* attribute), 247
am_pm_border_width (*kivymd.uix.pickers.timepicker.timepicker.MDTimePicker* attribute), 246
am_pm_radius (*kivymd.uix.pickers.timepicker.timepicker.MDTimePicker* attribute), 246
anchor (*kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect* attribute), 458
anchor (*kivymd.uix.button.button.MDFloatingActionButtonSpeedDial* attribute), 113
anchor (*kivymd.uix.card.card.MDCardSwipe* attribute), 208
anchor (*kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer* attribute), 328
anchor (*kivymd.uix.navigationrail.navigationrail.MDNavigationRail* attribute), 170
anchor_title (*kivymd.uix.backdrop.backdrop.MDBackdrop* attribute), 136
anchor_title (*kivymd.uix.toolbar.toolbar.MDTopAppBar* attribute), 153
angle (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior* attribute), 425
angle (*kivymd.uix.behaviors.elevation.CommonElevationBehavior* attribute), 446
anim_complete() (*kivymd.uix.behaviors.ripple_behavior.CommonRipple* method), 429
anim_duration (*kivymd.uix.tab.tab.MDTabs* attribute), 300
anim_rect() (*kivymd.uix.textfield.textfield.MDTextFieldRect* method), 338
anim_threshold (*kivymd.uix.tab.tab.MDTabs* attribute), 300
animate_opacity_icon() (*kivymd.uix.backdrop.backdrop.MDBackdrop* method), 138
animated_hero_in() (*kivymd.uix.transition.transition.MDTransitionBase* method), 381
animated_hero_out() (*kivymd.uix.transition.transition.MDTransitionBase* method), 381
animation (*kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet* attribute), 123
animation_display_banner() (*kivymd.uix.banner.banner.MDBanner* method), 196
animation_duration (*kivymd.uix.pickers.timepicker.timepicker.MDTimePicker* attribute), 247
animation_label() (*kivymd.uix.button.button.MDTextButton* method), 113
animation_segment_switch() (*kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl* method), 361
animation_size_ripple_area() (*kivymd.uix.navigationrail.navigationrail.MDNavigationRailItem* method), 169
animation_tooltip_dismiss() (*kivymd.uix.tooltip.tooltip.MDTooltip* method), 191
animation_tooltip_show() (*kivymd.uix.tooltip.tooltip.MDTooltip* method), 191
animation_transition (*kivymd.uix.pickers.timepicker.timepicker.MDTimePicker* attribute), 247
apply_state() (*kivymd.tools.hotreload.app.MDApp* method), 487
arguments (*kivymd.tools.hotreload.app.MDApp* property), 487
ArgumentParserWithHelp (class in *kivymd.tools.argument_parser*), 483
auto_dismiss (*kivymd.uix.snackbar.snackbar.BaseSnackbar* attribute), 308
AUTORELOADER_IGNORE_PATTERNS (*kivymd.tools.hotreload.app.MDApp* attribute), 486
AUTORELOADER_PATHS (*kivymd.tools.hotreload.app.MDApp* attribute), 486
back() (*kivymd.uix.filemanager.filemanager.MDFileManager* method), 378
back_color (*kivymd.uix.progressbar.progressbar.MDProgressBar* attribute), 188
back_layer_color (*kivymd.uix.backdrop.backdrop.MDBackdrop* attribute), 137
background (*kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior* attribute), 424
background (*kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet* attribute), 122
background_color (*kivymd.uix.datatables.datatables.MDDataTable* attribute), 90

background_color (kivymd.uix.menu.menu.MDDropdownMenu attribute), 275
background_color (kivymd.uix.sliverappbar.sliverappbar.MDSLiverAppBar attribute), 131
background_color (kivymd.uix.tab.tab.MDTabs attribute), 300
background_color_cell (kivymd.uix.datatables.datatables.MDDDataTable attribute), 91
background_color_header (kivymd.uix.datatables.datatables.MDDDataTable attribute), 91
background_color_selected_cell (kivymd.uix.datatables.datatables.MDDDataTable attribute), 92
background_down (kivymd.uix.behaviors.toggle_behavior.MDToggleBehavior attribute), 421
background_down_button_selected_type_color (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 240
background_hue (kivymd.uix.behaviors.backgroundcolor_behavior.MDBackgroundColorBehavior attribute), 425
background_normal (kivymd.uix.behaviors.toggle_behavior.MDToggleBehavior attribute), 421
background_origin (kivymd.uix.behaviors.backgroundcolor_behavior.MDBackgroundColorBehavior attribute), 425
background_palette (kivymd.uix.behaviors.backgroundcolor_behavior.MDBackgroundColorBehavior attribute), 425
BackgroundColorBehavior (class in kivymd.uix.behaviors.backgroundcolor_behavior), 424
badge_bg_color (kivymd.uix.label.label.MDIcon attribute), 258
badge_bg_color (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 167
badge_font_size (kivymd.uix.label.label.MDIcon attribute), 258
badge_font_size (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 168
badge_icon (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar attribute), 385
badge_icon (kivymd.uix.label.label.MDIcon attribute), 258
badge_icon (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 165
badge_icon_color (kivymd.uix.label.label.MDIcon attribute), 258
badge_icon_color (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 166
BaseButton (class in kivymd.uix.button.button), 109
BaseDialog (class in kivymd.uix.dialog.dialog), 364
BaseDialogPicker (class in kivymd.uix.pickers.datepicker.datepicker), 222
BaseListItem (class in kivymd.uix.list.list), 405
BaseSnackbar (class in kivymd.uix.snackbar.snackbar), 368
bg_color (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 123
bg_color (kivymd.uix.list.list.BaseListItem attribute), 406
bg_color (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 308
bg_color_root_button (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
bg_color_stack_button (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
bg_darkest (kivymd.theming.ThemeManager attribute), 18
bg_highest (kivymd.theming.ThemeManager attribute), 19
bg_normal (kivymd.theming.ThemeManager attribute), 19
bind() (kivymd.utils.asynckivy.event method), 507
bind_key() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 487
body (kivymd.uix.dialog.dialog.MDDialog attribute), 366
body (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 366
border_margin (kivymd.uix.menu.menu.MDDropdownMenu attribute), 272
build() (kivymd.tools.hotreload.app.MDApp method), 486
build_app() (kivymd.tools.hotreload.app.MDApp method), 487
buttons (kivymd.uix.dialog.dialog.MDDialog attribute), 366
buttons (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 366
call_ripple_animation_methods() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 429
callback (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 114
callback() (kivymd.utils.asynckivy.event method), 507
caller (kivymd.uix.menu.menu.MDDropdownMenu attribute), 275
can_capitalize (kivymd.uix.label.label.MDLabel attribute), 257
cancel_all_animations_on_double_click() (kivymd.uix.textfield.textfield.MDTextField method), 344

cancelable (kivymd.uix.taptargetview.MDTapTargetView attribute), 55
cancelable (kivymd.uix.taptargetview.MDTapTargetView attribute), 49
caption (kivymd.uix.bottomsheet.bottomsheet.GridBottomSheetItem method), 138
caption (kivymd.uix.bottomsheet.bottomsheet.GridBottomSheetItem attribute), 125
catching_duration (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 378
catching_duration (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
catching_transition (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
catching_up() (kivymd.uix.progressbar.progressbar.MDProgressBar method), 189
change_month() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
change_month() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 237
check (kivymd.uix.datatables.datatables.MDDataTable attribute), 88
check_content() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 325
check_determinate() (kivymd.uix.spinner.spinner.MDSpinner method), 281
check_font_styles() (kivymd.uix.label.label.MDLabel method), 257
check_open_panel() (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
check_overflow_cls() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 153
check_position_caller() (kivymd.uix.menu.menu.MDDropdownMenu method), 277
check_size() (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
check_size() (kivymd.uix.progressbar.progressbar.MDProgressBar method), 188
check_text() (kivymd.uix.textfield.textfield.MDTextField method), 345
check_transition() (kivymd.uix.screenmanager.MDScreenManager method), 57
checkbox_icon_down (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox attribute), 350
checkbox_icon_down (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox attribute), 350
checkbox_icon_normal (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox attribute), 349
checkbox_icon_normal (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox attribute), 349
CheckboxLeftWidget (class in kivymd.uix.list.list), 409
circular_padding (kivymd.uix.circularlayout.MDCircularLayout attribute), 55
circular_radius (kivymd.uix.circularlayout.MDCircularLayout attribute), 55
CircularElevationBehavior (class in kivymd.uix.behaviors.elevation), 450
CircularRippleBehavior (class in kivymd.uix.behaviors.ripple_behavior), 429
CLASSES (kivymd.tools.hotreload.app.MDApp attribute), 486
clockwise (kivymd.uix.circularlayout.MDCircularLayout attribute), 55

281
 color_active (kivymd.uix.selectioncontrol.selectioncontrol.MDColorSelectedItem attribute), 350
 color_disabled (kivymd.uix.button.button.MDTextButton attribute), 113
 color_icon_root_button (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
 color_icon_stack_button (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
 color_inactive (kivymd.uix.selectioncontrol.selectioncontrol.MDColorSelectedItem attribute), 350
 colors (in module kivymd.color_definitions), 27
 column_data (kivymd.uix.datatables.datatables.MDDDataTable attribute), 83
 command() (in module kivymd.tools.release.git_commands), 498
 CommonElevationBehavior (class in kivymd.uix.behaviors.elevation), 445
 CommonRipple (class in kivymd.uix.behaviors.ripple_behavior), 427
 compare_date_range() (kivymd.uix.pickers.datepicker.datepicker.MDDDatePicker method), 237
 complete_swipe() (kivymd.uix.card.card.MDCardSwipe method), 210
 content (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel attribute), 251
 content_cls (kivymd.uix.dialog.dialog.MDDialog attribute), 371
 create_argument_parser() (in module kivymd.tools.release.make_release), 499
 create_buttons() (kivymd.uix.dialog.dialog.MDDialog method), 374
 create_clock() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 419
 create_items() (kivymd.uix.dialog.dialog.MDDialog method), 373
 create_pagination_menu() (kivymd.uix.datatables.datatables.MDDDataTable method), 95
 create_unreleased_changelog() (in module kivymd.tools.release.make_release), 499
 current (kivymd.uix.bottomnavigation.bottomnavigation.TabbedPanelBase attribute), 385
 current_active_segment (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 361
 current_hero (kivymd.uix.screenmanager.MDScreenManager attribute), 57
 current_item (kivymd.uix.dropdownitem.dropdownitem.MDDropDownItem attribute), 380
 current_path (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 current_selected_item (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 181
 D
 data (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 114
 DataBase (class in kivymd.tools.patterns.MVC.Model.database_firebase), 446
 DataBase (class in kivymd.tools.patterns.MVC.Model.database_restdb), 446
 datas (in module kivymd.tools.packaging.pyinstaller.hook-kivymd), 489
 date_range_text_error (kivymd.uix.pickers.datepicker.datepicker.MDDDatePicker attribute), 236
 DatePickerInputField (class in kivymd.uix.pickers.datepicker.datepicker), 233
 day (kivymd.uix.pickers.datepicker.datepicker.MDDDatePicker attribute), 235
 DEBUG (kivymd.tools.hotreload.app.MDApp attribute), 486
 DeclarativeBehavior (class in kivymd.uix.behaviors.declarative_bahavior), 435
 default_color (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 240
 degree_spacing (kivymd.uix.circularlayout.MDCircularLayout attribute), 55
 delete_clock() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 419
 delete_clock() (kivymd.uix.tooltip.tooltip.MDTooltip method), 191
 delete_doc_from_collection() (kivymd.tools.patterns.MVC.Model.database_restdb.DataBase method), 497
 description_text (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
 description_text_bold (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
 description_text_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
 description_text_size (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
 deselect_item() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
 desktop_view (kivymd.uix.responsivelayout.MDResponsiveLayout attribute), 53

- `detect_visible` (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 423
- `determinate` (kivymd.uix.spinner.spinner.MDSpinner attribute), 280
- `determinate_time` (kivymd.uix.spinner.spinner.MDSpinner attribute), 281
- `DEVICE_IOS` (in module kivymd.material_resources), 478
- `device_ios` (kivymd.theming.ThemeableBehavior attribute), 24
- `device_orientation` (kivymd.theming.ThemeManager attribute), 20
- `DEVICE_TYPE` (in module kivymd.material_resources), 478
- `disabled_color` (kivymd.uix.button.button.BaseButton attribute), 111
- `disabled_color` (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox attribute), 351
- `disabled_hint_text_color` (kivymd.theming.ThemeManager attribute), 20
- `disabled_primary_color` (kivymd.theming.ThemeManager attribute), 19
- `dismiss()` (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet method), 124
- `dismiss()` (kivymd.uix.menu.menu.MDDropdownMenu method), 278
- `dismiss()` (kivymd.uix.snackbar.snackbar.BaseSnackbar method), 309
- `displacement` (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
- `display_tooltip()` (kivymd.uix.tooltip.tooltip.MDTooltip method), 191
- `divider` (kivymd.uix.list.list.BaseListItem attribute), 406
- `divider_color` (kivymd.theming.ThemeManager attribute), 19
- `divider_color` (kivymd.uix.list.list.BaseListItem attribute), 406
- `do_animation_check()` (kivymd.uix.chip.chip.MDChip method), 416
- `do_animation_open_stack()` (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
- `do_layout()` (kivymd.uix.circularlayout.MDCircularLayout method), 55
- `download_file()` (in module kivymd.tools.release.update_icons), 500
- `dp` (in module kivymd.material_resources), 478
- `drag_threshold` (kivymd.effects.roulettescroll.roulettescroll.MDRouletteScrollEffect attribute), 457
- `drag_threshold` (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
- `draw_shadow` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 449
- `draw_shadow` (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
- `duration` (kivymd.toast.kivytoast.kivytoast.Toast attribute), 482
- `duration` (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 308
- `duration_closing` (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 122
- `duration_long_touch` (kivymd.uix.behaviors.touch_behavior.TouchBehavior attribute), 419
- `duration_opening` (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 122
- ## E
- `edge_bottom` (kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect attribute), 456
- `edge_top` (kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect attribute), 456
- `edit_data()` (kivymd.tools.patterns.MVC.Model.database_restdb.Database method), 497
- `edit_padding_for_item()` (kivymd.uix.dialog.dialog.MDDialog method), 373
- `effect_cls` (kivymd.uix.datatables.datatables.MDDDataTable attribute), 92
- `elevation` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 445
- `elevation` (kivymd.uix.card.card.MDCard attribute), 207
- `elevation` (kivymd.uix.datatables.datatables.MDDDataTable attribute), 88
- `elevation` (kivymd.uix.menu.menu.MDDropdownMenu attribute), 277
- `elevation` (kivymd.uix.tab.tab.MDTabs attribute), 300
- `enable_autoreload()` (kivymd.tools.hotreload.app.MDApp method), 487
- `enable_swiping` (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 331
- `enter_point` (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 423
- `error` (kivymd.uix.textfield.textfield.MDTextField attribute), 340
- `error()` (kivymd.tools.argument_parser.ArgumentParserWithHelp method), 484
- `error_color` (kivymd.theming.ThemeManager attribute), 20
- `error_color` (kivymd.uix.textfield.textfield.MDTextField attribute), 339
- `eventclass` (class in kivymd.utils.asyncivy), 507
- `ExceptionClass` (class in kivymd.tools.hotreload.app), 489
- `exit_manager` (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377

export_icon_definitions() (in module `kivymd.tools.release.update_icons`), 500
ext (`kivymd.uix.filemanager.filemanager.MDFileManager` attribute), 377
F
fade_color (`kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect` attribute), 456
fade_height (`kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect` attribute), 456
fade_in() (`kivymd.toast.kivytoast.kivytoast.Toast` method), 482
fade_out() (`kivymd.toast.kivytoast.kivytoast.Toast` method), 482
fade_out() (`kivymd.uix.behaviors.ripple_behavior.CommonRipple` method), 429
FadingEdgeEffect (class in `kivymd.effects.fadingedge.fadingedge`), 456
FakeCircularElevationBehavior (class in `kivymd.uix.behaviors.elevation`), 451
FakeRectangularElevationBehavior (class in `kivymd.uix.behaviors.elevation`), 450
fbind() (`kivymd.tools.patterns.MVC.libs.translation.Translation` method), 497
fill_color_focus (`kivymd.uix.textfield.textfield.MDTextField` attribute), 339
fill_color_normal (`kivymd.uix.textfield.textfield.MDTextField` attribute), 339
finish_ripple() (`kivymd.uix.behaviors.ripple_behavior.CommonRipple` method), 429
first_widget (`kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation` attribute), 387
FitImage (class in `kivymd.uix.fitimage.fitimage`), 312
fixed_tab_label_width (`kivymd.uix.tab.tab.MDTabs` attribute), 300
focus_behavior (`kivymd.uix.behaviors.focus_behavior.FocusBehavior` attribute), 453
focus_behavior (`kivymd.uix.card.card.MDCard` attribute), 207
focus_color (`kivymd.uix.behaviors.focus_behavior.FocusBehavior` attribute), 453
FocusBehavior (class in `kivymd.uix.behaviors.focus_behavior`), 453
font_color_down (`kivymd.uix.behaviors.toggle_behavior.MDToggleBehavior` attribute), 421
font_color_normal (`kivymd.uix.behaviors.toggle_behavior.MDToggleBehavior` attribute), 421
font_name (`kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation` attribute), 387
font_name (`kivymd.uix.button.button.BaseButton` attribute), 110
font_name (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` attribute), 182
font_name (`kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker` attribute), 232
font_name (`kivymd.uix.tab.tab.MDTabs` attribute), 301
font_name_helper_text (`kivymd.uix.textfield.textfield.MDTextField` attribute), 344
font_name_hint_text (`kivymd.uix.textfield.textfield.MDTextField` attribute), 344
font_name_max_length (`kivymd.uix.textfield.textfield.MDTextField` attribute), 344
font_path (in module `kivymd.tools.release.update_icons`), 499
font_size (`kivymd.uix.button.button.BaseButton` attribute), 110
font_size (`kivymd.uix.dropdownitem.dropdownitem.MDDropDownItem` attribute), 380
font_size (`kivymd.uix.snackbar.snackbar.Snackbar` attribute), 309
font_size (`kivymd.uix.textfield.textfield.MDTextField` attribute), 343
font_style (`kivymd.uix.button.button.BaseButton` attribute), 110
font_style (`kivymd.uix.label.label.MDLabel` attribute), 257
font_style (`kivymd.uix.list.list.BaseListItem` attribute), 405
font_styles (`kivymd.theming.ThemeManager` attribute), 21
font_version (in module `kivymd.tools.release.update_icons`), 499
fonts (in module `kivymd.font_definitions`), 32
fonts_path (in module `kivymd`), 477
force_shadow_pos() (`kivymd.uix.behaviors.elevation.CommonElevationBehavior` method), 449
force_title_icon_mode (`kivymd.uix.tab.tab.MDTabs` attribute), 301
FOREGROUND_LOCK (`kivymd.tools.hotreload.app.MDApp` attribute), 486
format_help() (`kivymd.tools.argument_parser.ArgumentParserWithHelp` method), 484
FpsMonitor (class in `kivymd.utils.fpsmonitor`), 508
front_layer_color (`kivymd.uix.backdrop.backdrop.MDBackdrop` attribute), 137
funbind() (`kivymd.tools.patterns.MVC.libs.translation.Translation` method), 497
G
generate_list_widgets_days() (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` method), 237
generate_list_widgets_years() (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` method), 237

method), 237
 get_access_string() (kivymd.uix.filemanager.filemanager.MDFileManager method), 378
 get_angle() (kivymd.uix.circularlayout.MDCircularLayout method), 55
 get_color_instruction() (kivymd.uix.textfield.textfield.MDTextFieldRect method), 338
 get_connect() (in module kivymd.tools.patterns.MVC.Model.database_firebase), 496
 get_connect() (in module kivymd.tools.patterns.MVC.Model.database_restdb), 497
 get_content() (kivymd.uix.filemanager.filemanager.MDFileManager method), 378
 get_contrast_text_color() (in module kivymd.theming_dynamic_text), 478
 get_current_index() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 get_current_item() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 get_current_tab() (kivymd.uix.tab.tab.MDTabs method), 301
 get_data_from_collection() (kivymd.tools.patterns.MVC.Model.database_firebase method), 496
 get_data_from_collection() (kivymd.tools.patterns.MVC.Model.database_restdb method), 497
 get_date_range() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
 get_default_overflow_cls() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 get_default_toolbar() (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 133
 get_dist_from_side() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 331
 get_field() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
 get_hero_from_widget() (kivymd.uix.screenmanager.MDScreenManager method), 57
 get_hook_dirs() (in module kivymd.tools.packaging.pyinstaller), 489
 get_icons_list() (in module kivymd.tools.release.update_icons), 500
 get_items() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
 get_items() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 get_list_date() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 234
 get_normal_height() (kivymd.uix.dialog.dialog.MDDialog method), 373
 get_previous_version() (in module kivymd.tools.release.git_commands), 498
 get_pyinstaller_tests() (in module kivymd.tools.packaging.pyinstaller), 489
 get_real_device_type() (kivymd.uix.controllers.windowcontroller.WindowController method), 417
 get_rect_instruction() (kivymd.uix.textfield.textfield.MDTextFieldRect method), 338
 get_rgb() (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
 get_root() (kivymd.tools.hotreload.app.MDApp method), 486
 get_root_path() (kivymd.tools.hotreload.app.MDApp method), 487
 get_row_checks() (kivymd.uix.datatables.datatables.MDDDataTable method), 95
 get_selected() (kivymd.uix.selection.selection.MDSelectionList method), 216
 get_selected_list_items() (kivymd.uix.selection.selection.MDSelectionList method), 216
 get_tabs() (kivymd.uix.tab.tab.MDTabs method), 301
 get_md_date_picker() (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
 get_state() (kivymd.uix.pickers.timepicker.timepicker.MDTimePicker method), 247
 get_tab_list() (kivymd.uix.tab.tab.MDTabs method), 301
 get_term_vel() (kivymd.effects.roulettescroll.roulettescroll.RouletteScroll method), 458
 get_window_width_resizing_direction() (kivymd.uix.controllers.windowcontroller.WindowController method), 417
 git_clean() (in module kivymd.tools.release.git_commands), 498
 git_commit() (in module kivymd.tools.release.git_commands), 498
 git_push() (in module kivymd.tools.release.git_commands), 498
 git_tag() (in module kivymd.tools.release.git_commands), 498
 GridBottomSheetItem (class in kivymd.uix.bottomsheet.bottomsheet), 125
 grow() (kivymd.uix.behaviors.magic_behavior.MagicBehavior

method), 437

H

- `halign` (kivymd.uix.button.button.BaseButton attribute), 109
- `handle_exception()` (kivymd.tools.hotreload.app.ExceptionClass method), 485
- `hard_shadow_cl` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
- `hard_shadow_offset` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
- `hard_shadow_pos` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
- `hard_shadow_size` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
- `hard_shadow_texture` (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
- `header` (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
- `header` (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationItem attribute), 385
- `header_cls` (kivymd.uix.menu.menu.MDDropdownMenu attribute), 268
- `header_text` (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
- `headline_text` (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 153
- `headline_text_color` (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 153
- `helper_text` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 234
- `helper_text` (kivymd.uix.textfield.textfield.MDTextField attribute), 338
- `helper_text_color_focus` (kivymd.uix.textfield.textfield.MDTextField attribute), 340
- `helper_text_color_normal` (kivymd.uix.textfield.textfield.MDTextField attribute), 340
- `helper_text_mode` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 233
- `helper_text_mode` (kivymd.uix.textfield.textfield.MDTextField attribute), 338
- `hero_from_widget` (kivymd.uix.transition.transition.MDTransitionBase attribute), 381
- `hero_to` (kivymd.uix.screen.MDScreen attribute), 56
- `hero_widget` (kivymd.uix.transition.transition.MDTransitionBase attribute), 381
- `hide()` (kivymd.uix.banner.banner.MDBanner method), 196
- `hide_toolbar` (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 132
- `hint` (kivymd.uix.slider.slider.MDSlider attribute), 96
- `hint_animation` (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 116
- `hint_bg_color` (kivymd.uix.slider.slider.MDSlider attribute), 97
- `hint_radius` (kivymd.uix.slider.slider.MDSlider attribute), 97
- `hint_text_color` (kivymd.uix.slider.slider.MDSlider attribute), 97
- `hint_text_color_focus` (kivymd.uix.textfield.textfield.MDTextField attribute), 340
- `hint_text_color_normal` (kivymd.uix.textfield.textfield.MDTextField attribute), 340
- `hooks_path` (in module kivymd.tools.packaging.pyinstaller), 489
- `hor_growth` (kivymd.uix.menu.menu.MDDropdownMenu attribute), 274
- `horizontal_margins` (kivymd.theming.ThemeManager attribute), 21
- `hour` (kivymd.uix.pickers.timepicker.timepicker.MDTimePicker attribute), 244
- `hour_radius` (kivymd.uix.pickers.timepicker.timepicker.MDTimePicker attribute), 245
- `hover_visible` (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 423
- `HoverBehavior` (class in kivymd.uix.behaviors.hover_behavior), 423
- `hovering` (kivymd.uix.behaviors.hover_behavior.HoverBehavior attribute), 423
- `hue` (in module kivymd.color_definitions), 29
- I**
- `icon` (kivymd.uix.banner.banner.MDBanner attribute), 195
- `icon` (kivymd.uix.bottomnavigation.bottomnavigation.MDTab attribute), 385
- `icon` (kivymd.uix.button.button.BaseButton attribute), 110
- `icon` (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 112
- `icon` (kivymd.uix.button.button.MDIconButton attribute), 112
- `icon` (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel attribute), 251
- `icon` (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
- `icon` (kivymd.uix.label.label.MDIcon attribute), 258
- `icon` (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 326
- `icon` (kivymd.uix.navigationrail.navigationrail.MDNavigationRailFlatButton attribute), 162

- `icon` (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` attribute), 163
- `icon` (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` attribute), 162
- `icon` (`kivymd.uix.selection.selection.MDSelectionList` attribute), 215
- `icon` (`kivymd.uix.tab.tab.MDTabsBase` attribute), 298
- `icon` (`kivymd.uix.toolbar.toolbar.MDTopAppBar` attribute), 152
- `icon_active` (`kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch` attribute), 352
- `icon_active_color` (`kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch` attribute), 353
- `icon_bg_color` (`kivymd.uix.selection.selection.MDSelectionList` attribute), 216
- `icon_check_color` (`kivymd.uix.chip.chip.MDChip` attribute), 416
- `icon_check_color` (`kivymd.uix.selection.selection.MDSelectionList` attribute), 216
- `icon_color` (`kivymd.theming.ThemeManager` attribute), 20
- `icon_color` (`kivymd.uix.button.button.BaseButton` attribute), 110
- `icon_color` (`kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerList` attribute), 326
- `icon_color` (`kivymd.uix.toolbar.toolbar.MDTopAppBar` attribute), 153
- `icon_color_item_active` (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` attribute), 178
- `icon_color_item_normal` (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` attribute), 177
- `icon_definitions_path` (in module `kivymd.tools.release.update_icons`), 499
- `icon_folder` (`kivymd.uix.filemanager.filemanager.MDFileManager` attribute), 377
- `icon_inactive` (`kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch` attribute), 352
- `icon_inactive_color` (`kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch` attribute), 353
- `icon_left` (`kivymd.uix.chip.chip.MDChip` attribute), 415
- `icon_left` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 342
- `icon_left_color` (`kivymd.uix.chip.chip.MDChip` attribute), 416
- `icon_left_color_focus` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 341
- `icon_left_color_normal` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 341
- `icon_right` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 342
- `icon_right_color` (`kivymd.uix.chip.chip.MDChip` attribute), 416
- `icon_right_color_focus` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 341
- `icon_right_color_normal` (`kivymd.uix.textfield.textfield.MDTextField` attribute), 341
- `icon_size` (`kivymd.uix.bottomsheet.bottomsheet.GridBottomSheetItem` attribute), 125
- `icon_size` (`kivymd.uix.button.button.BaseButton` attribute), 110
- `IconLeftWidget` (class in `kivymd.uix.list.list`), 409
- `IconLeftWidgetWithoutTouch` (class in `kivymd.uix.list.list`), 409
- `IconRightWidget` (class in `kivymd.uix.list.list`), 409
- `IconRightWidgetWithoutTouch` (class in `kivymd.uix.list.list`), 409
- `id` (`kivymd.uix.behaviors.declarative_behavior.DeclarativeBehavior` attribute), 435
- `IDLE_DETECTION` (`kivymd.tools.hotreload.app.MDApp` attribute), 486
- `IDLE_TIMEOUT` (`kivymd.tools.hotreload.app.MDApp` attribute), 486
- `ILeftBodyTouch` (class in `kivymd.uix.list.list`), 407
- `ImageLeftWidget` (class in `kivymd.uix.list.list`), 408
- `ImageLeftWidgetWithoutTouch` (class in `kivymd.uix.list.list`), 408
- `ImageRightWidget` (class in `kivymd.uix.list.list`), 408
- `ImageRightWidgetWithoutTouch` (class in `kivymd.uix.list.list`), 408
- `in_md5_path` (in module `kivymd`), 477
- `indicator_color` (`kivymd.uix.tab.tab.MDTabs` attribute), 300
- `input_field_background_color` (`kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker` attribute), 230
- `input_field_cls` (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` attribute), 236
- `input_field_text_color` (`kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker` attribute), 231
- `input_filter()` (`kivymd.uix.pickers.datepicker.datepicker.DatePickerInput` method), 233
- `install_idle()` (`kivymd.tools.hotreload.app.MDApp` method), 487
- `interval` (`kivymd.effects.rouettescroll.rouettescroll.RouletteScrollEffect` attribute), 458

IRightBodyTouch (*class in kivymd.uix.list.list*), 407
 is_date_valaid() (*kivymd.uix.pickers.datepicker.datepicker.MDDDatePicker method*), 236
 is_numeric() (*kivymd.uix.pickers.datepicker.datepicker.MDDDatePicker method*), 234
 items (*kivymd.uix.dialog.dialog.MDDialog attribute*), 367
 items (*kivymd.uix.menu.menu.MDDropdownMenu attribute*), 269
 items_spacing (*kivymd.uix.swiper.swiper.MDSwiper attribute*), 392

K
 kivymd
 module, 477
 kivymd.app
 module, 25
 kivymd.color_definitions
 module, 27
 kivymd.effects
 module, 478
 kivymd.effects.fadingedge
 module, 479
 kivymd.effects.fadingedge.fadingedge
 module, 455
 kivymd.effects.roulettescroll
 module, 479
 kivymd.effects.roulettescroll.roulettescroll
 module, 456
 kivymd.effects.stiffscroll
 module, 479
 kivymd.effects.stiffscroll.stiffscroll
 module, 453
 kivymd.factory_registers
 module, 477
 kivymd.font_definitions
 module, 32
 kivymd.icon_definitions
 module, 30
 kivymd.material_resources
 module, 478
 kivymd.theming
 module, 7
 kivymd.theming_dynamic_text
 module, 478
 kivymd.toast
 module, 479
 kivymd.toast.androidtoast
 module, 479
 kivymd.toast.androidtoast.androidtoast
 module, 480
 kivymd.toast.kivytoast
 module, 481
 kivymd.toast.kivytoast.kivytoast
 module, 481
 kivyMDTools
 module, 483
 kivyMDTools.Fildgment_parser
 module, 483
 kivyMDTools.hotreload
 module, 484
 kivyMDTools.hotreload.app
 module, 484
 kivyMDTools.packaging
 module, 488
 kivyMDTools.packaging.pyinstaller
 module, 488
 kivyMDTools.packaging.pyinstaller.hook-kivymd
 module, 489
 kivyMDTools.patterns
 module, 489
 kivyMDTools.patterns.add_view
 module, 489
 kivyMDTools.patterns.create_project
 module, 490
 kivyMDTools.patterns.MVC
 module, 496
 kivyMDTools.patterns.MVC.libs
 module, 497
 kivyMDTools.patterns.MVC.libs.translation
 module, 497
 kivyMDTools.patterns.MVC.Model
 module, 496
 kivyMDTools.patterns.MVC.Model.database_firebase
 module, 496
 kivyMDTools.patterns.MVC.Model.database_restdb
 module, 496
 kivyMDTools.release
 module, 498
 kivyMDTools.release.git_commands
 module, 498
 kivyMDTools.release.make_release
 module, 498
 kivyMDTools.release.update_icons
 module, 499
 kivymd.uix
 module, 500
 kivymd.uix.anchorlayout
 module, 33
 kivymd.uix.backdrop
 module, 501
 kivymd.uix.backdrop.backdrop
 module, 134
 kivymd.uix.banner
 module, 501
 kivymd.uix.banner.banner
 module, 192
 kivymd.uix.behaviors

module, 501	module, 502
kivymd.uix.behaviors.backgroundcolor_behavior	kivymd.uix.dialog.dialog
module, 423	module, 362
kivymd.uix.behaviors.declarative_bahavior	kivymd.uix.dropdownitem
module, 429	module, 502
kivymd.uix.behaviors.elevation	kivymd.uix.dropdownitem.dropdownitem
module, 438	module, 378
kivymd.uix.behaviors.focus_behavior	kivymd.uix.expansionpanel
module, 451	module, 502
kivymd.uix.behaviors.hover_behavior	kivymd.uix.expansionpanel.expansionpanel
module, 421	module, 248
kivymd.uix.behaviors.magic_behavior	kivymd.uix.filemanager
module, 436	module, 503
kivymd.uix.behaviors.ripple_behavior	kivymd.uix.filemanager.filemanager
module, 425	module, 374
kivymd.uix.behaviors.toggle_behavior	kivymd.uix.fitimage
module, 419	module, 503
kivymd.uix.behaviors.touch_behavior	kivymd.uix.fitimage.fitimage
module, 417	module, 310
kivymd.uix.bottomnavigation	kivymd.uix.floatlayout
module, 501	module, 74
kivymd.uix.bottomnavigation.bottomnavigation	kivymd.uix.gridlayout
module, 381	module, 71
kivymd.uix.bottomsheet	kivymd.uix.hero
module, 501	module, 62
kivymd.uix.bottomsheet.bottomsheet	kivymd.uix.imagelist
module, 117	module, 503
kivymd.uix.boxlayout	kivymd.uix.imagelist.imagelist
module, 57	module, 75
kivymd.uix.button	kivymd.uix.label
module, 501	module, 503
kivymd.uix.button.button	kivymd.uix.label.label
module, 100	module, 252
kivymd.uix.card	kivymd.uix.list
module, 502	module, 503
kivymd.uix.card.card	kivymd.uix.list.list
module, 197	module, 395
kivymd.uix.carousel	kivymd.uix.menu
module, 72	module, 503
kivymd.uix.chip	kivymd.uix.menu.menu
module, 502	module, 258
kivymd.uix.chip.chip	kivymd.uix.navigationdrawer
module, 409	module, 503
kivymd.uix.circularlayout	kivymd.uix.navigationdrawer.navigationdrawer
module, 53	module, 313
kivymd.uix.controllers	kivymd.uix.navigationrail
module, 502	module, 503
kivymd.uix.controllers.windowcontroller	kivymd.uix.navigationrail.navigationrail
module, 416	module, 155
kivymd.uix.datatables	kivymd.uix.pickers
module, 502	module, 504
kivymd.uix.datatables.datatables	kivymd.uix.pickers.colorpicker
module, 79	module, 504
kivymd.uix.dialog	kivymd.uix.pickers.colorpicker.colorpicker

module, 237	module, 303
kivymd.uix.pickers.datepicker	kivymd.uix.spinner
module, 504	module, 505
kivymd.uix.pickers.datepicker.datepicker	kivymd.uix.spinner.spinner
module, 217	module, 278
kivymd.uix.pickers.timepicker	kivymd.uix.stacklayout
module, 504	module, 59
kivymd.uix.pickers.timepicker.timepicker	kivymd.uix.swiper
module, 241	module, 505
kivymd.uix.progressbar	kivymd.uix.swiper.swiper
module, 504	module, 389
kivymd.uix.progressbar.progressbar	kivymd.uix.tab
module, 184	module, 505
kivymd.uix.recyclegridlayout	kivymd.uix.tab.tab
module, 35	module, 281
kivymd.uix.recycleview	kivymd.uix.taptargetview
module, 59	module, 37
kivymd.uix.refreshlayout	kivymd.uix.templates
module, 504	module, 506
kivymd.uix.refreshlayout.refreshlayout	kivymd.uix.templates.rotatewidget
module, 77	module, 506
kivymd.uix.relativelayout	kivymd.uix.templates.rotatewidget.rotatewidget
module, 61	module, 459
kivymd.uix.responsivelayout	kivymd.uix.templates.scalewidget
module, 51	module, 506
kivymd.uix.screen	kivymd.uix.templates.scalewidget.scalewidget
module, 56	module, 461
kivymd.uix.screenmanager	kivymd.uix.templates.stencilwidget
module, 57	module, 506
kivymd.uix.scrollview	kivymd.uix.templates.stencilwidget.stencilwidget
module, 50	module, 463
kivymd.uix.segmentedcontrol	kivymd.uix.textfield
module, 504	module, 506
kivymd.uix.segmentedcontrol.segmentedcontrol	kivymd.uix.textfield.textfield
module, 357	module, 332
kivymd.uix.selection	kivymd.uix.toolbar
module, 505	module, 506
kivymd.uix.selection.selection	kivymd.uix.toolbar.toolbar
module, 210	module, 139
kivymd.uix.selectioncontrol	kivymd.uix.tooltip
module, 505	module, 506
kivymd.uix.selectioncontrol.selectioncontrol	kivymd.uix.tooltip.tooltip
module, 346	module, 189
kivymd.uix.slider	kivymd.uix.transition
module, 505	module, 507
kivymd.uix.slider.slider	kivymd.uix.transition.transition
module, 96	module, 380
kivymd.uix.sliverappbar	kivymd.uix.widget
module, 505	module, 34
kivymd.uix.sliverappbar.sliverappbar	kivymd.utils
module, 126	module, 507
kivymd.uix.snackbar	kivymd.utils.asynckivy
module, 505	module, 507
kivymd.uix.snackbar.snackbar	kivymd.utils.fpsmonitor

- module, 507
- kivymd.utils.setBarsColors
 - module, 508
- kivymd_path (in module kivymd.tools.release.update_icons), 499
- KV_DIRS (kivymd.tools.hotreload.app.MDApp attribute), 486
- KV_FILES (kivymd.tools.hotreload.app.MDApp attribute), 486
- L**
- label_check_texture_size() (kivymd.toast.kivytoast.kivytoast.Toast method), 482
- label_text_color (kivymd.uix.button.button.MDFloatingActionButton attribute), 114
- lay_canvas_instructions() (kivymd.uix.behaviors.ripple_behavior.CircularRippleBehavior method), 429
- lay_canvas_instructions() (kivymd.uix.behaviors.ripple_behavior.CommonRippleBehavior method), 429
- lay_canvas_instructions() (kivymd.uix.behaviors.ripple_behavior.RectangularRippleBehavior method), 429
- left_action (kivymd.uix.banner.banner.MDBanner attribute), 196
- left_action_items (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
- left_action_items (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 147
- light_colors (in module kivymd.color_definitions), 29
- line_anim (kivymd.uix.textfield.textfield.MDTextField attribute), 339
- line_anim (kivymd.uix.textfield.textfield.MDTextFieldRect attribute), 337
- line_color (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute), 424
- line_color (kivymd.uix.button.button.BaseButton attribute), 110
- line_color_disabled (kivymd.uix.button.button.BaseButton attribute), 110
- line_color_focus (kivymd.uix.textfield.textfield.MDTextField attribute), 339
- line_color_normal (kivymd.uix.textfield.textfield.MDTextField attribute), 339
- line_width (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute), 424
- line_width (kivymd.uix.button.button.BaseButton attribute), 110
- line_width (kivymd.uix.spinner.spinner.MDSpinner attribute), 281
- load_all_kv_files() (kivymd.app.MDApp method), 27
- load_app_dependencies() (kivymd.tools.hotreload.app.MDApp method), 487
- lock_swiping (kivymd.uix.tab.tab.MDTabs attribute), 301
- M**
- magic_speed (kivymd.uix.behaviors.magic_behavior.MagicBehavior attribute), 437
- MagicBehavior (class in kivymd.uix.behaviors.magic_behavior), 437
- main() (in module kivymd.tools.patterns.add_view), 490
- main() (in module kivymd.tools.release.make_release), 499
- main() (in module kivymd.tools.release.update_icons), 500
- make_icon_definitions() (in module kivymd.tools.release.update_icons), 500
- material_style (kivymd.theming.ThemeManager attribute), 16
- max (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect attribute), 457
- max (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
- max_date (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 236
- max_degree (kivymd.uix.circularlayout.MDCircularLayout attribute), 55
- max_friction (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
- max_height (kivymd.uix.menu.menu.MDDropdownMenu attribute), 271
- max_height (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 132
- max_height (kivymd.uix.textfield.textfield.MDTextField attribute), 343
- max_length_text_color (kivymd.uix.textfield.textfield.MDTextField attribute), 342
- MAX_NAV_DRAWER_WIDTH (in module kivymd.material_resources), 478
- max_opacity (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 133
- max_opened_x (kivymd.uix.card.card.MDCardSwipe attribute), 209
- max_swipe_x (kivymd.uix.card.card.MDCardSwipe attribute), 209
- max_text_length (kivymd.uix.textfield.textfield.MDTextField attribute), 339

max_year (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 235
 md_bg_bottom_color (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 151
 md_bg_color (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute), 424
 md_bg_color (kivymd.uix.button.button.BaseButton attribute), 111
 md_bg_color (kivymd.uix.dialog.dialog.MDDialog attribute), 373
 md_bg_color (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 359
 md_bg_color (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 129
 md_bg_color (kivymd.uix.toolbar.toolbar.MDBottomAppBar attribute), 155
 md_bg_color_disabled (kivymd.uix.button.button.BaseButton attribute), 111
 md_icons (in module kivymd.icon_definitions), 32
 MDAdaptiveWidget (class in kivymd.uix), 500
 MDAnchorLayout (class in kivymd.uix.anchorlayout), 34
 MDApp (class in kivymd.app), 26
 MDApp (class in kivymd.tools.hotreload.app), 485
 MDBackdrop (class in kivymd.uix.backdrop.backdrop), 136
 MDBackdropBackLayer (class in kivymd.uix.backdrop.backdrop), 139
 MDBackdropFrontLayer (class in kivymd.uix.backdrop.backdrop), 139
 MDBackdropToolbar (class in kivymd.uix.backdrop.backdrop), 139
 MDBanner (class in kivymd.uix.banner.banner), 195
 MDBottomAppBar (class in kivymd.uix.toolbar.toolbar), 155
 MDBottomNavigation (class in kivymd.uix.bottomnavigation.bottomnavigation), 386
 MDBottomNavigationItem (class in kivymd.uix.bottomnavigation.bottomnavigation), 385
 MDBottomSheet (class in kivymd.uix.bottomsheet.bottomsheet), 122
 MDBoxLayout (class in kivymd.uix.boxlayout), 59
 MDCard (class in kivymd.uix.card.card), 207
 MDCardSwipe (class in kivymd.uix.card.card), 208
 MDCardSwipeFrontBox (class in kivymd.uix.card.card), 210
 MDCardSwipeLayerBox (class in kivymd.uix.card.card), 210
 MDCarousel (class in kivymd.uix.carousel), 73
 MDCheckbox (class in kivymd.uix.selectioncontrol.selectioncontrol), 349
 MDChip (class in kivymd.uix.chip.chip), 415
 MDCircularLayout (class in kivymd.uix.circularlayout), 55
 MDColorPicker (class in kivymd.uix.pickers.colorpicker.colorpicker), 346
 MDCustomBottomSheet (class in kivymd.uix.bottomsheet.bottomsheet), 124
 MDDataTable (class in kivymd.uix.datatables.datatables), 80
 MDDatePicker (class in kivymd.uix.pickers.datepicker.datepicker), 234
 MDDialog (class in kivymd.uix.dialog.dialog), 364
 MDDropDownItem (class in kivymd.uix.dropdownitem.dropdownitem), 379
 MDDropdownMenu (class in kivymd.uix.menu.menu), 268
 MDExpansionPanel (class in kivymd.uix.expansionpanel.expansionpanel), 250
 MDExpansionPanelLabel (class in kivymd.uix.expansionpanel.expansionpanel), 250
 MDExpansionPanelOneLine (class in kivymd.uix.expansionpanel.expansionpanel), 250
 MDExpansionPanelThreeLine (class in kivymd.uix.expansionpanel.expansionpanel), 250
 MDExpansionPanelTwoLine (class in kivymd.uix.expansionpanel.expansionpanel), 250
 MDFadeSlideTransition (class in kivymd.uix.transition.transition), 381
 MDFileManager (class in kivymd.uix.filemanager.filemanager), 377
 MDFillRoundFlatButton (class in kivymd.uix.button.button), 112
 MDFillRoundFlatIconButton (class in kivymd.uix.button.button), 112
 MDFlatButton (class in kivymd.uix.button.button), 111
 MDFloatingActionButton (class in kivymd.uix.button.button), 112
 MDFloatingActionButtonSpeedDial (class in kivymd.uix.button.button), 113
 MDFloatLayout (class in kivymd.uix.floatlayout), 74
 MDGridBottomSheet (class in kivymd.uix.bottomsheet.bottomsheet), 126
 MDGridLayout (class in kivymd.uix.gridlayout), 72
 MDHeroFrom (class in kivymd.uix.hero), 70
 MDHeroTo (class in kivymd.uix.hero), 71
 MDIcon (class in kivymd.uix.label.label), 258
 MDIconButton (class in kivymd.uix.button.button), 112
 MDLabel (class in kivymd.uix.label.label), 257

MDList (class in <i>kivymd.uix.list.list</i>), 404	<i>kivymd.uix.button.button</i>), 112
MDListBottomSheet (class in <i>kivymd.uix.bottomsheet.bottomsheet</i>), 124	MDScreen (class in <i>kivymd.uix.screen</i>), 56
MDNavigationDrawer (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 327	MDScreenManager (class in <i>kivymd.uix.screenmanager</i>), 57
MDNavigationDrawerDivider (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 321	MDScrollView (class in <i>kivymd.uix.scrollview</i>), 50
MDNavigationDrawerHeader (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 322	MDScrollViewRefreshLayout (class in <i>kivymd.uix.refreshlayout.refreshlayout</i>), 79
MDNavigationDrawerItem (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 325	MDSegmentedControl (class in <i>kivymd.uix.segmentedcontrol.segmentedcontrol</i>), 359
MDNavigationDrawerLabel (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 320	MDSegmentedControlItem (class in <i>kivymd.uix.segmentedcontrol.segmentedcontrol</i>), 359
MDNavigationDrawerMenu (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 326	MDSelectionList (class in <i>kivymd.uix.selection.selection</i>), 215
MDNavigationLayout (class in <i>kivymd.uix.navigationdrawer.navigationdrawer</i>), 320	MDSeparator (class in <i>kivymd.uix.card.card</i>), 207
MDNavigationRail (class in <i>kivymd.uix.navigationrail.navigationrail</i>), 169	MDSlider (class in <i>kivymd.uix.slider.slider</i>), 96
MDNavigationRailFabButton (class in <i>kivymd.uix.navigationrail.navigationrail</i>), 162	MDSlideTransition (class in <i>kivymd.uix.transition.transition</i>), 381
MDNavigationRailItem (class in <i>kivymd.uix.navigationrail.navigationrail</i>), 163	MDSliverAppBar (class in <i>kivymd.uix.sliverappbar.sliverappbar</i>), 129
MDNavigationRailMenuButton (class in <i>kivymd.uix.navigationrail.navigationrail</i>), 162	MDSliverAppBarContent (class in <i>kivymd.uix.sliverappbar.sliverappbar</i>), 129
MDProgressBar (class in <i>kivymd.uix.progressbar.progressbar</i>), 188	MDSliverAppBarHeader (class in <i>kivymd.uix.sliverappbar.sliverappbar</i>), 129
MDRaisedButton (class in <i>kivymd.uix.button.button</i>), 112	MDSpinner (class in <i>kivymd.uix.spinner.spinner</i>), 280
MDRectangleFlatButton (class in <i>kivymd.uix.button.button</i>), 112	MDStackLayout (class in <i>kivymd.uix.stacklayout</i>), 61
MDRectangleFlatIconButton (class in <i>kivymd.uix.button.button</i>), 112	MDSwapTransition (class in <i>kivymd.uix.transition.transition</i>), 381
MDRecycleGridLayout (class in <i>kivymd.uix.recyclegridlayout</i>), 36	MDSwiper (class in <i>kivymd.uix.swiper.swiper</i>), 392
MDRecycleView (class in <i>kivymd.uix.recycleview</i>), 59	MDSwiperItem (class in <i>kivymd.uix.swiper.swiper</i>), 392
MDRelativeLayout (class in <i>kivymd.uix.relativelayout</i>), 61	MDSwitch (class in <i>kivymd.uix.selectioncontrol.selectioncontrol</i>), 352
MDResponsiveLayout (class in <i>kivymd.uix.responsivelayout</i>), 52	MDTab (class in <i>kivymd.uix.bottomnavigation.bottomnavigation</i>), 385
MDRoundFlatButton (class in <i>kivymd.uix.button.button</i>), 112	MDTabs (class in <i>kivymd.uix.tab.tab</i>), 299
MDRoundFlatIconButton (class in <i>kivymd.uix.button.button</i>), 112	MDTabsBase (class in <i>kivymd.uix.tab.tab</i>), 298
	MDTapTargetView (class in <i>kivymd.uix.taptargetview</i>), 46
	MDTextButton (class in <i>kivymd.uix.button.button</i>), 113
	MDTextField (class in <i>kivymd.uix.textfield.textfield</i>), 338
	MDTextFieldRect (class in <i>kivymd.uix.textfield.textfield</i>), 337
	MDTimePicker (class in <i>kivymd.uix.pickers.timepicker.timepicker</i>), 244
	MDToggleButton (class in <i>kivymd.uix.behaviors.toggle_behavior</i>), 421
	MDTooltip (class in <i>kivymd.uix.tooltip.tooltip</i>), 190
	MDTooltipViewClass (class in <i>kivymd.uix.tooltip.tooltip</i>), 192
	MDTopAppBar (class in <i>kivymd.uix.toolbar.toolbar</i>), 147
	MDTransitionBase (class in <i>kivymd.uix.transition.transition</i>), 381

MDWidget (class in `kivymd.uix.widget`), 35
 min (`kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect` attribute), 457
 min (`kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect` attribute), 454
 min_date (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` attribute), 235
 min_year (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` attribute), 235
 minute (`kivymd.uix.pickers.timepicker.timepicker.MDTimePicker` attribute), 244
 minute_radius (`kivymd.uix.pickers.timepicker.timepicker.MDTimePicker` attribute), 244
 mipmap (`kivymd.uix.fitimage.fitimage.FitImage` attribute), 312
 mobile_view (`kivymd.uix.responsivelayout.MDResponsiveLayout` attribute), 52
 mode (`kivymd.uix.pickers.datepicker.datepicker.MDDatePicker` attribute), 235
 mode (`kivymd.uix.textfield.textfield.MDTextField` attribute), 339
 mode (`kivymd.uix.toolbar.toolbar.MDTopAppBar` attribute), 149
 module
 kivymd, 477
 kivymd.app, 25
 kivymd.color_definitions, 27
 kivymd.effects, 478
 kivymd.effects.fadingedge, 479
 kivymd.effects.fadingedge.fadingedge, 455
 kivymd.effects.roulettescroll, 479
 kivymd.effects.roulettescroll.roulettescroll, 456
 kivymd.effects.stiffscroll, 479
 kivymd.effects.stiffscroll.stiffscroll, 453
 kivymd.factory_registers, 477
 kivymd.font_definitions, 32
 kivymd.icon_definitions, 30
 kivymd.material_resources, 478
 kivymd.theming, 7
 kivymd.theming_dynamic_text, 478
 kivymd.toast, 479
 kivymd.toast.androidtoast, 479
 kivymd.toast.androidtoast.androidtoast, 480
 kivymd.toast.kivytoast, 481
 kivymd.toast.kivytoast.kivytoast, 481
 kivymd.tools, 483
 kivymd.tools.argument_parser, 483
 kivymd.tools.hotreload, 484
 kivymd.tools.hotreload.app, 484
 kivymd.tools.packaging, 488
 kivymd.tools.packaging.pyinstaller, 488
 kivymd.tools.packaging.pyinstaller.hook-kivymd, 489
 kivymd.tools.patterns, 489
 kivymd.tools.patterns.add_view, 489
 kivymd.tools.patterns.create_project, 490
 kivymd.tools.patterns.MVC, 496
 kivymd.tools.patterns.MVC.libs, 497
 kivymd.tools.patterns.MVC.libs.translation, 497
 kivymd.tools.patterns.MVC.Model, 496
 kivymd.tools.patterns.MVC.Model.database_firebase, 496
 kivymd.tools.patterns.MVC.Model.database_restdb, 496
 kivymd.tools.release, 498
 kivymd.tools.release.git_commands, 498
 kivymd.tools.release.make_release, 498
 kivymd.tools.release.update_icons, 499
 kivymd.uix, 500
 kivymd.uix.anchorlayout, 33
 kivymd.uix.backdrop, 501
 kivymd.uix.backdrop.backdrop, 134
 kivymd.uix.banner, 501
 kivymd.uix.banner.banner, 192
 kivymd.uix.behaviors, 501
 kivymd.uix.behaviors.backgroundcolor_behavior, 423
 kivymd.uix.behaviors.declarative_bahavior, 429
 kivymd.uix.behaviors.elevation, 438
 kivymd.uix.behaviors.focus_behavior, 451
 kivymd.uix.behaviors.hover_behavior, 421
 kivymd.uix.behaviors.magic_behavior, 436
 kivymd.uix.behaviors.ripple_behavior, 425
 kivymd.uix.behaviors.toggle_behavior, 419
 kivymd.uix.behaviors.touch_behavior, 417
 kivymd.uix.bottomnavigation, 501
 kivymd.uix.bottomnavigation.bottomnavigation, 381
 kivymd.uix.bottomsheet, 501
 kivymd.uix.bottomsheet.bottomsheet, 117
 kivymd.uix.boxlayout, 57
 kivymd.uix.button, 501
 kivymd.uix.button.button, 100
 kivymd.uix.card, 502
 kivymd.uix.card.card, 197
 kivymd.uix.carousel, 72
 kivymd.uix.chip, 502
 kivymd.uix.chip.chip, 409
 kivymd.uix.circularlayout, 53
 kivymd.uix.controllers, 502
 kivymd.uix.controllers.windowcontroller, 416
 kivymd.uix.datatables, 502

- kivymd.uix.datatables.datatables, 79
 - kivymd.uix.dialog, 502
 - kivymd.uix.dialog.dialog, 362
 - kivymd.uix.dropdownitem, 502
 - kivymd.uix.dropdownitem.dropdownitem, 378
 - kivymd.uix.expansionpanel, 502
 - kivymd.uix.expansionpanel.expansionpanel, 248
 - kivymd.uix.filemanager, 503
 - kivymd.uix.filemanager.filemanager, 374
 - kivymd.uix.fitimage, 503
 - kivymd.uix.fitimage.fitimage, 310
 - kivymd.uix.floatlayout, 74
 - kivymd.uix.gridlayout, 71
 - kivymd.uix.hero, 62
 - kivymd.uix.imagelist, 503
 - kivymd.uix.imagelist.imagelist, 75
 - kivymd.uix.label, 503
 - kivymd.uix.label.label, 252
 - kivymd.uix.list, 503
 - kivymd.uix.list.list, 395
 - kivymd.uix.menu, 503
 - kivymd.uix.menu.menu, 258
 - kivymd.uix.navigationdrawer, 503
 - kivymd.uix.navigationdrawer.navigationdrawer, 313
 - kivymd.uix.navigationrail, 503
 - kivymd.uix.navigationrail.navigationrail, 155
 - kivymd.uix.pickers, 504
 - kivymd.uix.pickers.colorpicker, 504
 - kivymd.uix.pickers.colorpicker.colorpicker, 237
 - kivymd.uix.pickers.datepicker, 504
 - kivymd.uix.pickers.datepicker.datepicker, 217
 - kivymd.uix.pickers.timepicker, 504
 - kivymd.uix.pickers.timepicker.timepicker, 241
 - kivymd.uix.progressbar, 504
 - kivymd.uix.progressbar.progressbar, 184
 - kivymd.uix.recyclegridlayout, 35
 - kivymd.uix.recycleview, 59
 - kivymd.uix.refreshlayout, 504
 - kivymd.uix.refreshlayout.refreshlayout, 77
 - kivymd.uix.relativelayout, 61
 - kivymd.uix.responsivelayout, 51
 - kivymd.uix.screen, 56
 - kivymd.uix.screenmanager, 57
 - kivymd.uix.scrollview, 50
 - kivymd.uix.segmentedcontrol, 504
 - kivymd.uix.segmentedcontrol.segmentedcontrol, 357
 - kivymd.uix.selection, 505
 - kivymd.uix.selection.selection, 210
 - kivymd.uix.selectioncontrol, 505
 - kivymd.uix.selectioncontrol.selectioncontrol, 346
 - kivymd.uix.slider, 505
 - kivymd.uix.slider.slider, 96
 - kivymd.uix.sliverappbar, 505
 - kivymd.uix.sliverappbar.sliverappbar, 126
 - kivymd.uix.snackbar, 505
 - kivymd.uix.snackbar.snackbar, 303
 - kivymd.uix.spinner, 505
 - kivymd.uix.spinner.spinner, 278
 - kivymd.uix.stacklayout, 59
 - kivymd.uix.swiper, 505
 - kivymd.uix.swiper.swiper, 389
 - kivymd.uix.tab, 505
 - kivymd.uix.tab.tab, 281
 - kivymd.uix.taptargetview, 37
 - kivymd.uix.templates, 506
 - kivymd.uix.templates.rotatewidget, 506
 - kivymd.uix.templates.rotatewidget.rotatewidget, 459
 - kivymd.uix.templates.scalewidget, 506
 - kivymd.uix.templates.scalewidget.scalewidget, 461
 - kivymd.uix.templates.stencilwidget, 506
 - kivymd.uix.templates.stencilwidget.stencilwidget, 463
 - kivymd.uix.textfield, 506
 - kivymd.uix.textfield.textfield, 332
 - kivymd.uix.toolbar, 506
 - kivymd.uix.toolbar.toolbar, 139
 - kivymd.uix.tooltip, 506
 - kivymd.uix.tooltip.tooltip, 189
 - kivymd.uix.transition, 507
 - kivymd.uix.transition.transition, 380
 - kivymd.uix.widget, 34
 - kivymd.utils, 507
 - kivymd.utils.asynckivy, 507
 - kivymd.utils.fpsmonitor, 507
 - kivymd.utils.setBarsColors, 508
 - monotonic (in module *kivymd.tools.hotreload.app*), 485
 - month (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker* attribute), 235
 - move_changelog() (in module *kivymd.tools.release.make_release*), 499
- ## N
- name (*kivymd.tools.patterns.MVC.Model.database_firebase.DataBase* attribute), 496
 - name (*kivymd.tools.patterns.MVC.Model.database_restdb.DataBase* attribute), 497

navigation_rail (kivymd.uix.navigationrail.navigationrail.MDNavigationRailItem
 attribute), 163
 near_next_notch() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect.
 method), 458
 near_notch() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect.
 method), 458
 nearest_notch() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect.
 method), 458
 next_notch() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect.
 method), 458
 no_ripple_effect (kivymd.uix.tab.tab.MDTabs attribute), 301
O
 ObservableShadow (class in kivymd.uix.behaviors.elevation), 450
 observers (kivymd.tools.patterns.MVC.libs.translation.Translation attribute), 497
 on__is_off() (kivymd.uix.slider.slider.MDSlider method), 100
 on__rotation_angle() (kivymd.uix.spinner.spinner.MDSpinner method), 281
 on__shadow_pos() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 450
 on_action_button() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_active() (kivymd.uix.chip.chip.MDChip method), 416
 on_active() (kivymd.uix.navigationrail.navigationrail.MDNavigationRailItem method), 169
 on_active() (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl method), 362
 on_active() (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox method), 352
 on_active() (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch method), 357
 on_active() (kivymd.uix.slider.slider.MDSlider method), 100
 on_active() (kivymd.uix.spinner.spinner.MDSpinner method), 281
 on_adaptive_height() (kivymd.uix.MDAdaptiveWidget method), 500
 on_adaptive_size() (kivymd.uix.MDAdaptiveWidget method), 501
 on_adaptive_width() (kivymd.uix.MDAdaptiveWidget method), 500
 on_anchor() (kivymd.uix.card.card.MDCardSwipe method), 210
 on_anchor_title() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_background_color() (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 116
 on_background_down_button_selected_type_color() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 241
 on_bg_color_stack_button() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_bg_hint_color() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_buttons() (kivymd.uix.snackbar.snackbar.BaseSnackbar method), 309
 on_cancel() (kivymd.uix.pickers.datepicker.datepicker.BaseDatePicker method), 233
 on_carousel_index() (kivymd.uix.tab.tab.MDTabs method), 302
 on_change_screen_type() (kivymd.uix.responsivelayout.MDResponsiveLayout method), 53
 on_check_press() (kivymd.uix.datatables.datatables.MDDDataTable method), 95
 on_close() (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
 on_close() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_close() (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
 on_close() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_coasted_to_stop() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect method), 459
 on_color_icon_root_button() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_color_icon_stack_button() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_complete() (kivymd.uix.transition.transition.MDTransitionBase method), 381
 on_data() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_description_text() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_description_text_bold() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_description_text_size() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_determinate_complete() (kivymd.uix.taptargetview.MDTapTargetView method), 49

(kivymd.uix.spinner.spinner.MDSpinner method), 281
 on_device_orientation() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 236
 on_disabled() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 450
 on_disabled() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView method), 385
 on_disabled() (kivymd.uix.button.button.BaseButton method), 111
 on_disabled() (kivymd.uix.button.button.MDTextButton method), 113
 on_disabled() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_dismiss() (kivymd.uix.menu.menu.MDDropdownMenu method), 278
 on_dismiss() (kivymd.uix.snackbar.snackbar.BaseSnackbar method), 309
 on_dismiss() (kivymd.uix.tooltip.tooltip.MDTooltip method), 192
 on_double_tap() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 419
 on_draw_shadow() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_elevation() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 450
 on_enter() (kivymd.uix.behaviors.focus_behavior.FocusBehavior method), 453
 on_enter() (kivymd.uix.behaviors.hover_behavior.HoverBehavior method), 423
 on_enter() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_enter() (kivymd.uix.tooltip.tooltip.MDTooltip method), 191
 on_error() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_focus() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_font_name() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView method), 387
 on_header() (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
 on_header_cls() (kivymd.uix.menu.menu.MDDropdownMenu method), 278
 on_height() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_helper_text() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_helper_text_color_normal() (kivymd.uix.textfield.textfield.MDTextField method), 346
 on_hero_to() (kivymd.uix.screen.MDScreen method), 56
 on_hint() (kivymd.uix.slider.slider.MDSlider method), 99
 on_hint_animation() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_hint_text_color() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_hint_text_color_normal() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_icon() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_icon() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_icon_color() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_icon_left() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_icon_right() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_icon_right_color_normal() (kivymd.uix.textfield.textfield.MDTextField method), 346
 on_idle() (kivymd.tools.hotreload.app.MDApp method), 487
 on_item_press() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
 on_item_release() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
 on_label_text_color() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_leave() (kivymd.uix.behaviors.focus_behavior.FocusBehavior method), 453
 on_leave() (kivymd.uix.behaviors.hover_behavior.HoverBehavior method), 423
 on_leave() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView method), 385
 on_leave() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_leave() (kivymd.uix.tooltip.tooltip.MDTooltip method), 192
 on_left_action_items() (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
 on_left_action_items() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_line_color_normal() (kivymd.uix.textfield.textfield.MDTextField method), 346
 on_long_touch() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 419
 on_long_touch() (kivymd.uix.chip.chip.MDChip

method), 416
 on_long_touch() (kivymd.uix.tooltip.tooltip.MDTooltip method), 191
 on_max_length_text_color() (kivymd.uix.textfield.textfield.MDTextField method), 346
 on_md_bg_bottom_color() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_md_bg_color() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_mode() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_mouse_update() (kivymd.uix.behaviors.hover_behavior.hover_behavior.MDHoverBehavior method), 423
 on_notch() (kivymd.effects.roulettescroll.roulettescroll.RoundRouletteScroll method), 458
 on_ok_button_pressed() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 236
 on_open() (kivymd.toast.kivytoast.kivytoast.Toast method), 482
 on_open() (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
 on_open() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 on_open() (kivymd.uix.dialog.dialog.MDDialog method), 373
 on_open() (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
 on_open() (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
 on_open() (kivymd.uix.snackbar.snackbar.BaseSnackbar method), 309
 on_open() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_open_progress() (kivymd.uix.card.card.MDCardSwipe method), 210
 on_opposite_colors() (kivymd.uix.label.label.MDLabel method), 257
 on_orientation() (kivymd.uix.card.card.MDSeparator method), 207
 on_outer_radius() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_outer_touch() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_outside_click() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_overflow_cls() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_overswipe_left() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_overswipe_right() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_palette() (kivymd.uix.spinner.spinner.MDSpinner method), 281
 on_pre_swipe() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_press() (kivymd.uix.button.button.MDTextButton method), 113
 on_press() (kivymd.uix.chip.chip.MDChip method), 416
 on_press() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 169
 on_pre_select() (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl method), 362
 on_progress_effect() (kivymd.uix.transition.transition.MDFadeSlideTransition method), 381
 on_radius() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 332
 on_ref_press() (kivymd.uix.tab.tab.MDTabs method), 302
 on_release() (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 169
 on_release() (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
 on_resize() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView method), 388
 on_right_action_items() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on Ripple behavior() (kivymd.uix.card.card.MDCard method), 208
 on_row_press() (kivymd.uix.datatables.datatables.MDDataTable method), 95
 on_save() (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker method), 233
 on_scroll_content() (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 133
 on_scroll_start() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_select_color() (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
 on_selected() (kivymd.uix.selection.selection.MDSelectionList method), 217
 on_selected_color_background() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView method), 387
 on_shadow_group() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 449
 on_shadow_pos() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 450
 on_show() (kivymd.uix.tooltip.tooltip.MDTooltip

method), 192
 on_show_off() (kivymd.uix.slider.slider.MDSlider method), 100
 on_size() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 388
 on_size() (kivymd.uix.controllers.windowcontroller.WindowController method), 417
 on_size() (kivymd.uix.responsivelayout.MDResponsiveLayout method), 53
 on_size() (kivymd.uix.tab.tab.MDTabs method), 302
 on_slide_complete() (kivymd.uix.carousel.MDCarousel method), 73
 on_slide_progress() (kivymd.uix.carousel.MDCarousel method), 73
 on_slide_progress() (kivymd.uix.tab.tab.MDTabs method), 302
 on_state() (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox method), 352
 on_swipe() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_swipe_complete() (kivymd.uix.card.card.MDCardSwipe method), 209
 on_swipe_left() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_swipe_right() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_switch_tabs() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 387
 on_switch_tabs() (kivymd.uix.pickers.colorpicker.ColorPicker method), 241
 on_tab_press() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 385
 on_tab_press() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 385
 on_tab_release() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 385
 on_tab_switch() (kivymd.uix.tab.tab.MDTabs method), 302
 on_tab_touch_down() (kivymd.uix.bottomnavigation.bottomnavigation.MDTab method), 385
 on_tab_touch_move() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 385
 on_tab_touch_up() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 385
 on_target_radius() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_target_touch() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_text() (kivymd.uix.dropdownitem.dropdownitem.MDDropdownMenu method), 380
 on_text_color() (kivymd.uix.label.label.MDLabel method), 257
 on_text_color_active() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 387
 on_text_color_normal() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation method), 387
 on_text_color_normal() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_theme_style() (kivymd.theming.ThemeManager method), 22
 on_theme_text_color() (kivymd.uix.label.label.MDLabel method), 257
 on_thumb_down() (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch method), 357
 on_title_text() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 on_title_text_bold() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_title_text_size() (kivymd.uix.taptargetview.MDTapTargetView method), 50
 on_toolbar_cls() (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 133
 on_touch_down() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 429
 on_touch_down() (kivymd.uix.button.button.BaseButton method), 111
 on_touch_down() (kivymd.uix.card.card.MDCardSwipe method), 210
 on_touch_down() (kivymd.uix.carousel.MDCarousel method), 73
 on_touch_down() (kivymd.uix.list.list.BaseListItem method), 406
 on_touch_down() (kivymd.uix.menu.menu.MDDropdownMenu method), 278
 on_touch_down() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 331
 on_touch_down() (kivymd.uix.slider.slider.MDSlider method), 100
 on_touch_down() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_touch_move() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 429
 on_touch_move() (kivymd.uix.card.card.MDCardSwipe method), 210
 on_touch_move() (kivymd.uix.list.list.BaseListItem method), 406
 on_touch_move() (kivymd.uix.menu.menu.MDDropdownMenu method), 278

method), 278
 on_touch_move() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 331
 on_touch_up() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 437
 on_touch_up() (kivymd.uix.behaviors.ripple_behavior.CircularRippleBehavior method), 429
 on_touch_up() (kivymd.uix.button.button.BaseButton method), 111
 on_touch_up() (kivymd.uix.card.card.MDCardSwipe method), 210
 on_touch_up() (kivymd.uix.carousel.MDCarousel method), 74
 on_touch_up() (kivymd.uix.list.list.BaseListItem method), 407
 on_touch_up() (kivymd.uix.menu.menu.MDDropdownMenu method), 278
 on_touch_up() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 332
 on_touch_up() (kivymd.uix.refreshlayout.refreshlayout.MDScrollViewRefreshLayout method), 79
 on_touch_up() (kivymd.uix.slider.slider.MDSlider method), 100
 on_touch_up() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 on_transform_in() (kivymd.uix.hero.MDHeroFrom method), 70
 on_transform_out() (kivymd.uix.hero.MDHeroFrom method), 71
 on_triple_tap() (kivymd.uix.behaviors.touch_behavior.TouchBehavior method), 419
 on_type() (kivymd.uix.button.button.MDFloatingActionButton method), 113
 on_type() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 332
 on_type() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 153
 on_type_color() (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
 on_type_height() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 on_unselected() (kivymd.uix.selection.selection.MDSelectionList method), 217
 on_use_text() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar method), 387
 on_value() (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect method), 454
 on_value_normalized() (kivymd.uix.slider.slider.MDSlider method), 99
 on_vbar() (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 133
 on_wakeup() (kivymd.tools.hotreload.app.MDApp method), 487
 on_width() (kivymd.uix.textfield.textfield.MDTextField method), 345
 on_width() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 153
 OneLineAvatarIconListItem (class in kivymd.uix.list.list), 408
 OneLineAvatarListItem (class in kivymd.uix.list.list), 408
 OneLineIconListItem (class in kivymd.uix.list.list), 408
 OneLineListItem (class in kivymd.uix.list.list), 408
 OneLineRightIconListItem (class in kivymd.uix.list.list), 408
 open() (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
 open() (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet method), 123
 open() (kivymd.uix.menu.menu.MDDropdownMenu method), 278
 open() (kivymd.uix.snackbar.snackbar.BaseSnackbar method), 109
 open_card() (kivymd.uix.card.card.MDCardSwipe method), 210
 open_panel() (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
 open_progress (kivymd.uix.card.card.MDCardSwipe attribute), 208
 open_progress (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 330
 open_stack() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 opening_time (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
 opening_time (kivymd.uix.banner.banner.MDBanner attribute), 196
 opening_time (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
 opening_time (kivymd.uix.card.card.MDCardSwipe attribute), 209
 opening_time (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel attribute), 251
 opening_time (kivymd.uix.menu.menu.MDDropdownMenu attribute), 275
 opening_time (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 332
 opening_time_button_rotation (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115
 opening_timeout (kivymd.uix.banner.banner.MDBanner attribute), 196
 opening_transition (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
 opening_transition (kivymd.uix.banner.banner.MDBanner attribute), 195
 opening_transition (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 115

- attribute*), 114
 - `opening_transition` (`kivymd.uix.card.card.MDCardSwipeOwner` *attribute*), 208
 - `opening_transition` (`kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel` *attribute*), 251
 - `opening_transition` (`kivymd.uix.menu.menu.MDDropdownMenu` *attribute*), 275
 - `opening_transition` (`kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer` *attribute*), 331
 - `opening_transition_button_rotation` (`kivymd.uix.button.button.MDFloatingActionButtonSpeedDial` *attribute*), 115
 - `opposite_bg_dark` (`kivymd.theming.ThemeManager` *attribute*), 18
 - `opposite_bg_darkest` (`kivymd.theming.ThemeManager` *attribute*), 18
 - `opposite_bg_light` (`kivymd.theming.ThemeManager` *attribute*), 19
 - `opposite_bg_normal` (`kivymd.theming.ThemeManager` *attribute*), 19
 - `opposite_colors` (`kivymd.theming.ThemableBehavior` *attribute*), 24
 - `opposite_colors` (`kivymd.uix.toolbar.toolbar.MDTopAppBar` *attribute*), 150
 - `opposite_disabled_hint_text_color` (`kivymd.theming.ThemeManager` *attribute*), 20
 - `opposite_disabled_primary_color` (`kivymd.theming.ThemeManager` *attribute*), 19
 - `opposite_divider_color` (`kivymd.theming.ThemeManager` *attribute*), 19
 - `opposite_icon_color` (`kivymd.theming.ThemeManager` *attribute*), 20
 - `opposite_secondary_text_color` (`kivymd.theming.ThemeManager` *attribute*), 20
 - `opposite_text_color` (`kivymd.theming.ThemeManager` *attribute*), 20
 - `orientation` (`kivymd.uix.progressbar.progressbar.MDProgressBar` *attribute*), 188
 - `original_argv` (in module `kivymd.tools.hotreload.app`), 485
 - `outer_circle_alpha` (`kivymd.uix.taptargetview.MDTapTargetView` *attribute*), 47
 - `outer_circle_color` (`kivymd.uix.taptargetview.MDTapTargetView` *attribute*), 46
 - `outer_radius` (`kivymd.uix.taptargetview.MDTapTargetView` *attribute*), 46
 - `over_widget` (`kivymd.uix.banner.banner.MDBanner` *attribute*), 195
 - `overflow_action_button_is_added` (`kivymd.uix.toolbar.toolbar.MDTopAppBar` *method*), 153
 - `overflow_cls` (`kivymd.uix.toolbar.toolbar.MDTopAppBar` *attribute*), 151
 - `overlay_color` (`kivymd.uix.selection.selection.MDSelectionList` *attribute*), 216
 - `owner` (`kivymd.uix.pickers.datepicker.datepicker.DatePickerInputField` *attribute*), 233
- ## P
- `padding` (`kivymd.uix.backdrop.backdrop.MDBackdrop` *attribute*), 136
 - `padding` (`kivymd.uix.button.button.MDFlatButton` *attribute*), 109
 - `padding` (`kivymd.uix.button.button.MDFloatingActionButtonSpeedDial` *attribute*), 111
 - `padding` (`kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer` *attribute*), 329
 - `padding` (`kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer` *attribute*), 322
 - `padding` (`kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer` *attribute*), 321
 - `padding` (`kivymd.uix.navigationrail.navigationrail.MDNavigationRail` *attribute*), 170
 - `pagination_menu_height` (`kivymd.uix.datatables.datatables.MDDDataTable` *attribute*), 89
 - `pagination_menu_pos` (`kivymd.uix.datatables.datatables.MDDDataTable` *attribute*), 89
 - `palette` (in module `kivymd.color_definitions`), 29
 - `palette` (`kivymd.uix.spinner.spinner.MDSpinner` *attribute*), 281
 - `panel_cls` (`kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel` *attribute*), 251
 - `panel_color` (`kivymd.uix.bottomnavigation.bottomnavigation.TabbedPanel` *attribute*), 385
 - `parent_background` (`kivymd.uix.label.label.MDLabel` *attribute*), 257
 - `parse_args` () (`kivymd.tools.argument_parser.ArgumentParserWithHelp` *method*), 484
 - `patch_builder` () (`kivymd.tools.hotreload.app.MDApp` *method*), 487
 - `path` (in module `kivymd`), 477
 - `pos_hint` (`kivymd.uix.list.list.IconLeftWidget` *attribute*), 409
 - `pos_hint` (`kivymd.uix.list.list.IconLeftWidgetWithoutTouch` *attribute*), 409
 - `pos_hint` (`kivymd.uix.list.list.IconRightWidget` *attribute*), 409
 - `pos_hint` (`kivymd.uix.list.list.IconRightWidgetWithoutTouch` *attribute*), 409
 - `position` (`kivymd.uix.menu.menu.MDDropdownMenu` *attribute*), 275
 - `prepare_foreground_lock` (`kivymd.tools.hotreload.app.MDApp` *method*), 487
 - `preview` (`kivymd.uix.filemanager.filemanager.MDFileManager` *attribute*), 377

previous_tab (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar attribute), 385
 primary_color (kivymd.theming.ThemeManager attribute), 13
 primary_color (kivymd.uix.pickers.datepicker.datepicker.BaseDatePicker attribute), 224
 primary_dark (kivymd.theming.ThemeManager attribute), 15
 primary_dark_hue (kivymd.theming.ThemeManager attribute), 13
 primary_hue (kivymd.theming.ThemeManager attribute), 12
 primary_light (kivymd.theming.ThemeManager attribute), 13
 primary_light_hue (kivymd.theming.ThemeManager attribute), 13
 primary_palette (kivymd.theming.ThemeManager attribute), 11
 progress_round_color (kivymd.uix.selection.selection.MDSelectionList attribute), 216
 progress_round_size (kivymd.uix.selection.selection.MDSelectionList attribute), 216
 propagate_touch_to_touchable_widgets() (kivymd.uix.list.list.BaseListItem method), 407
 pull_back_velocity (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect attribute), 458
 pull_duration (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect attribute), 458
 PY3 (in module kivymd.tools.hotreload.app), 485
R
 radio_icon_down (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox attribute), 350
 radio_icon_normal (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox attribute), 350
 radius (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior attribute), 424
 radius (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 447
 radius (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 123
 radius (kivymd.uix.card.card.MDCard attribute), 208
 radius (kivymd.uix.dialog.dialog.BaseDialog attribute), 364
 radius (kivymd.uix.list.list.BaseListItem attribute), 406
 radius (kivymd.uix.menu.menu.MDDropdownMenu attribute), 276
 radius (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 170
 radius (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker attribute), 223
 radius (kivymd.uix.picker.picker.MDPicker attribute), 360
 radius (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 132
 radius (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 309
 radius (kivymd.uix.templates.stencilwidget.stencilwidget.StencilWidget attribute), 465
 radius (kivymd.uix.textfield.textfield.MDTextField attribute), 344
 radius_color_scale (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 240
 radius_from (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 123
 radius_left (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
 radius_right (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 137
 RAISE_ERROR (kivymd.tools.hotreload.app.MDApp attribute), 486
 re_additional_icons (in module kivymd.tools.release.update_icons), 499
 re_icon_definitions (in module kivymd.tools.release.update_icons), 500
 re_icons_json (in module kivymd.tools.release.update_icons), 499
 re_quote_keys (in module kivymd.tools.release.update_icons), 500
 re_version (in module kivymd.tools.release.update_icons), 500
 re_version_in_file (in module kivymd.tools.release.update_icons), 500
 rearm_idle() (kivymd.tools.hotreload.app.MDApp method), 487
 rebuild() (kivymd.tools.hotreload.app.MDApp method), 487
 RectangularElevationBehavior (class in kivymd.uix.behaviors.elevation), 450
 RectangularRippleBehavior (class in kivymd.uix.behaviors.ripple_behavior), 429
 refresh_callback (kivymd.uix.refreshlayout.refreshlayout.MDScrollView attribute), 79
 refresh_done() (kivymd.uix.refreshlayout.refreshlayout.MDScrollViewRefreshLayout method), 79
 refresh_tabs() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar method), 387
 register (in module kivymd.factory_registers), 478
 release (in module kivymd), 477
 reload() (kivymd.uix.fitimage.fitimage.FitImage method), 312
 remove_notch() (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 154
 remove_overflow_button() (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 154

method), 153
 remove_row() (kivymd.uix.datatables.datatables.MDDDataTable attribute), 427
 method), 94
 remove_shadow() (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 180
 method), 154
 remove_tooltip() (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 301
 method), 191
 remove_widget() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationView attribute), 427
 method), 388
 remove_widget() (kivymd.uix.circularlayout.MDCircularLayout attribute), 428
 method), 55
 remove_widget() (kivymd.uix.list.list.BaseListItem attribute), 428
 method), 407
 remove_widget() (kivymd.uix.list.list.MDList attribute), 428
 method), 405
 remove_widget() (kivymd.uix.swiper.swiper.MDSwiper attribute), 428
 method), 393
 remove_widget() (kivymd.uix.tab.tab.MDTabs attribute), 428
 method), 302
 replace_in_file() (in module attribute), 427
 kivymd.tools.release.make_release), 499
 required (kivymd.uix.textfield.textfield.MDTextField attribute), 429
 attribute), 339
 reset_active_color() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 429
 method), 327
 resize_content_layout() (kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet attribute), 429
 method), 124
 return_action_button_to_toolbar() (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 429
 method), 153
 reversed (kivymd.uix.progressbar.progressbar.MDProgressbar attribute), 461
 attribute), 188
 right_action (kivymd.uix.banner.banner.MDBanner attribute), 461
 attribute), 196
 right_action_items (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 461
 attribute), 137
 right_action_items (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 461
 attribute), 148
 right_pad (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial attribute), 457
 attribute), 114
 right_text (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 461
 attribute), 326
 ripple_alpha (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 427
 attribute), 427
 ripple_behavior (kivymd.uix.card.card.MDCard attribute), 427
 attribute), 207
 ripple_canvas_after (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 428
 attribute), 428
 ripple_color (kivymd.theming.ThemeManager attribute), 20
 attribute), 20
 ripple_color (kivymd.uix.behaviors.ripple_behavior.CommonRipple attribute), 427
 attribute), 427

running_away() (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
 running_duration (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
 running_transition (kivymd.uix.progressbar.progressbar.MDProgressBar attribute), 188
S
 scale_value_x (kivymd.uix.templates.scalewidget.scalewidget.ScaleWidget attribute), 463
 scale_value_y (kivymd.uix.templates.scalewidget.scalewidget.ScaleWidget attribute), 463
 scale_value_z (kivymd.uix.templates.scalewidget.scalewidget.ScaleWidget attribute), 463
 ScaleWidget (class in kivymd.uix.templates.scalewidget.scalewidget), 463
 screen (kivymd.uix.bottomsheet.bottomsheet.MDCustomBottomSheet attribute), 124
 scrim_alpha_transition (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 331
 scrim_color (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 329
 scroll (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
 search (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 secondary_font_style (kivymd.uix.list.list.BaseListItem attribute), 406
 secondary_text (kivymd.uix.list.list.BaseListItem attribute), 405
 secondary_text_color (kivymd.theming.ThemeManager attribute), 20
 secondary_text_color (kivymd.uix.list.list.BaseListItem attribute), 405
 secondary_theme_text_color (kivymd.uix.list.list.BaseListItem attribute), 405
 segment_color (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 359
 segment_panel_height (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 360
 segment_switching_duration (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 361
 segment_switching_transition (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 361
 sel_day (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 236
 select_directory_on_press_button() (kivymd.uix.filemanager.filemanager.MDFileManager method), 378
 select_unpath (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 selected (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 326
 selected_all() (kivymd.uix.selection.selection.MDSelectionList method), 217
 selected_color (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 326
 selected_color (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 241
 selected_color (kivymd.uix.selectioncontrol.selectioncontrol.MDSelectionControl attribute), 351
 selected_color_background (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar attribute), 386
 selected_color_background (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 179
 selected_mode (kivymd.uix.selection.selection.MDSelectionList attribute), 215
 selection (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 378
 selector (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 378
 selector_color (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker attribute), 225
 separator_color (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl attribute), 360
 set__radius() (kivymd.uix.button.button.MDFloatingActionButton method), 113
 set_active_underline_color() (kivymd.uix.textfield.textfield.MDTextField method), 344
 set_active_underline_width() (kivymd.uix.textfield.textfield.MDTextField method), 344
 set_all_colors() (kivymd.uix.button.button.BaseButton method), 111
 setBars_color (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar attribute), 387
 setBars_color (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 151
 setBars_colors() (in module kivymd.utils.setBars_colors), 508

`set_bg_color()` (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar method), 129
`set_button_colors()` (kivymd.uix.button.button.BaseButton method), 111
`set_chevron_down()` (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
`set_chevron_up()` (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 251
`set_clearcolor` (kivymd.theming.ThemeManager attribute), 21
`set_clearcolor_by_theme_style()` (kivymd.theming.ThemeManager method), 22
`set_colors()` (kivymd.theming.ThemeManager method), 22
`set_colors_to_updated()` (kivymd.uix.textfield.textfield.MDTextField method), 344
`set_current()` (kivymd.uix.swiper.swiper.MDSwiper method), 394
`set_current_selected_item()` (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
`set_default_colors()` (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl method), 361
`set_default_colors()` (kivymd.uix.textfield.textfield.MDTextField method), 344
`set_disabled_color()` (kivymd.uix.button.button.BaseButton method), 111
`set_elevation()` (kivymd.uix.card.card.MDCard method), 208
`set_error()` (kivymd.tools.hotreload.app.MDApp method), 487
`set_error()` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 233
`set_fade()` (kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect method), 456
`set_fill_color()` (kivymd.uix.textfield.textfield.MDTextField method), 344
`set_font_size()` (kivymd.uix.button.button.MDFloatingActionButton method), 113
`set_headline_font_style()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 153
`set_helper_text_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_hint_text_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_hint_text_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_icon()` (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch method), 357
`set_icon_color()` (kivymd.uix.button.button.BaseButton method), 111
`set_icon_left_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_icon_right_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_item()` (kivymd.uix.dropdownitem.dropdownitem.MDDropDownItem method), 380
`set_line_color()` (kivymd.uix.card.card.MDCard method), 208
`set_max_length_text_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_max_text_length()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_md_bg_color()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`set_menu_properties()` (kivymd.uix.menu.menu.MDDropdownMenu method), 277
`set_month_day()` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
`set_new_icon()` (kivymd.uix.backdrop.backdrop.MDBackdrop method), 138
`set_notch()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`set_notch_rectangle()` (kivymd.uix.textfield.textfield.MDTextField method), 344
`set_placeholder_text_color()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_paddings()` (kivymd.uix.expansionpanel.expansionpanel.MDExpansionPanel method), 250
`set_pos_bottom_buttons()` (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
`set_pos_hint_text()` (kivymd.uix.textfield.textfield.MDTextField method), 345
`set_pos_labels()` (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
`set_pos_menu_fab_buttons()` (kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 184
`set_pos_panel_items()`

(kivymd.uix.navigationrail.navigationrail.MDNavigationRail method), 183
 set_pos_root_button() (kivymd.uix.button.button.MDFloatingActionButtonSpeedDial method), 116
 set_position_to_current_year() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
 set_radius() (kivymd.uix.button.button.BaseButton method), 111
 set_radius() (kivymd.uix.card.card.MDCard method), 208
 set_screen() (kivymd.uix.responsivelayout.MDResponsiveLayout method), 53
 set_selected_widget() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
 set_shadow() (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
 set_size() (kivymd.uix.button.button.MDFloatingActionButton method), 113
 set_size() (kivymd.uix.button.button.MDIconButton method), 112
 set_state() (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 331
 set_static_underline_color() (kivymd.uix.textfield.textfield.MDTextField method), 344
 set_status_bar_color() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar method), 387
 set_style() (kivymd.uix.card.card.MDCard method), 208
 set_term_vel() (kivymd.effects.roulletescroll.roulletescroll.RouletteScrollEffect method), 458
 set_text() (kivymd.uix.textfield.textfield.MDTextField method), 345
 set_text_color() (kivymd.uix.button.button.BaseButton method), 111
 set_text_full_date() (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
 set_thumb_icon() (kivymd.uix.slider.slider.MDSlider method), 99
 set_time() (kivymd.uix.pickers.timepicker.timepicker.MDTimePicker method), 247
 set_type_banner() (kivymd.uix.banner.banner.MDBanner method), 196
 set_widget() (kivymd.tools.hotreload.app.MDApp method), 487
 set_x_pos() (kivymd.uix.textfield.textfield.MDTextField method), 345
 shadow_group (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 447
 shadow_radius (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 447
 shadow_preset() (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 450
 shake() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 437
 sheet_list (kivymd.uix.bottomsheet.bottomsheet.MDListBottomSheet attribute), 125
 shift_left (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 191
 shift_right (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 191
 shift_size (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 191
 show() (kivymd.uix.banner.banner.MDBanner method), 196
 show() (kivymd.uix.filemanager.filemanager.MDFileManager method), 378
 show_disks() (kivymd.uix.filemanager.filemanager.MDFileManager method), 378
 show_hidden_files (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 show_off (kivymd.uix.slider.slider.MDSlider attribute), 392
 shrink() (kivymd.uix.behaviors.magic_behavior.MagicBehavior method), 437
 size_duration (kivymd.uix.swiper.swiper.MDSwiper attribute), 392
 size_transition (kivymd.uix.swiper.swiper.MDSwiper attribute), 392
 sleep() (in module kivymd.utils.asynckivy), 507
 Snackbar (class in kivymd.uix.snackbar.snackbar), 309
 snackbar_animation_dir (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 309
 snackbar_x (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 309
 snackbar_y (kivymd.uix.snackbar.snackbar.BaseSnackbar attribute), 309
 soft_shadow_cl (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
 soft_shadow_offset (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
 soft_shadow_pos (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 448
 soft_shadow_size (kivymd.uix.behaviors.elevation.CommonElevationBehavior attribute), 447
 sort_by (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 sort_by_desc (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
 sorted_on (kivymd.uix.datatables.datatables.MDDDataTable attribute), 87
 sorted_order (kivymd.uix.datatables.datatables.MDDDataTable attribute), 87

attribute), 88
 source (kivymd.uix.bottomsheet.bottomsheet.GridBottomSheet attribute), 125
 source (kivymd.uix.fitimage.fitimage.FitImage attribute), 312
 source (kivymd.uix.label.label.MDIcon attribute), 258
 source (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 323
 spacing (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 327
 specific_secondary_text_color (kivymd.uix.behaviors.backgroundcolor_behavior.SpecificBackgroundColorBehavior attribute), 425
 specific_text_color (kivymd.uix.behaviors.backgroundcolor_behavior.SpecificBackgroundColorBehavior attribute), 425
 SpecificBackgroundColorBehavior (class in kivymd.uix.behaviors.backgroundcolor_behavior), 425
 standard_increment (kivymd.theming.ThemeManager attribute), 21
 start() (in module kivymd.utils.asynckivy), 507
 start() (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect method), 458
 start() (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect method), 454
 start() (kivymd.uix.progressbar.progressbar.MDProgressBar method), 188
 start() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 start() (kivymd.uix.transition.transition.MDFadeSlideTransition method), 381
 start() (kivymd.uix.transition.transition.MDTransitionBase method), 381
 start() (kivymd.utils.fpsmonitor.FpsMonitor method), 508
 start_from (kivymd.uix.circularlayout.MDCircularLayout attribute), 55
 start_ripple() (kivymd.uix.behaviors.ripple_behavior.CommonRipple method), 429
 state (kivymd.uix.button.button.MDFloatingActionButton attribute), 115
 state (kivymd.uix.card.card.MDCardSwipe attribute), 209
 state (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 330
 state (kivymd.uix.taptargetview.MDTapTargetView attribute), 49
 status (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 330
 StencilWidget (class in kivymd.uix.templates.stencilwidget.stencilwidget), 465
 StiffScrollEffect (class in kivymd.effects.stiffscroll.stiffscroll), 454
 stop() (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect method), 455
 stop() (kivymd.uix.progressbar.progressbar.MDProgressBar method), 188
 stop() (kivymd.uix.taptargetview.MDTapTargetView method), 49
 stop_on_outer_touch (kivymd.uix.taptargetview.MDTapTargetView attribute), 49
 stop_on_target_touch (kivymd.uix.taptargetview.MDTapTargetView attribute), 49
 style (kivymd.uix.card.card.MDCard attribute), 208
 swipe_distance (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 331
 swipe_distance (kivymd.uix.swiper.swiper.MDSwiper attribute), 393
 swipe_edge_width (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 331
 swipe_effect() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 swipe_on_scroll (kivymd.uix.swiper.swiper.MDSwiper attribute), 393
 swipe_right() (kivymd.uix.swiper.swiper.MDSwiper method), 394
 swipe_transition (kivymd.uix.swiper.swiper.MDSwiper attribute), 392
 switch_lang() (kivymd.tools.patterns.MVC.libs.translation.Translation method), 497
 switch_tab() (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation attribute), 387
 switch_tab() (kivymd.uix.tab.tab.MDTabs method), 301
 sync_theme_styles() (kivymd.theming.ThemeManager method), 429
 tab_bar_height (kivymd.uix.tab.tab.MDTabs attribute), 299
 tab_header (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation attribute), 387
 tab_hint_x (kivymd.uix.tab.tab.MDTabs attribute), 300
 tab_indicator_anim (kivymd.uix.tab.tab.MDTabs attribute), 299
 tab_indicator_error_height (kivymd.uix.tab.tab.MDTabs attribute), 299
 tab_indicator_type (kivymd.uix.tab.tab.MDTabs attribute), 299
 tab_label (kivymd.uix.tab.tab.MDTabsBase attribute), 299

tab_label_font_style (kivymd.uix.tab.tab.MDTabsBase attribute), 299
 tab_label_text (kivymd.uix.tab.tab.MDTabsBase attribute), 299
 tab_padding (kivymd.uix.tab.tab.MDTabs attribute), 299
 TabbedPanelBase (class in kivymd.uix.bottomnavigation.bottomnavigation), 385
 tablet_view (kivymd.uix.responsivelayout.MDResponsiveLayout attribute), 52
 tabs (kivymd.uix.bottomnavigation.bottomnavigation.TabbedPanelBase attribute), 386
 tag (kivymd.uix.hero.MDHeroFrom attribute), 70
 target_circle_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 47
 target_radius (kivymd.uix.taptargetview.MDTapTargetView attribute), 47
 target_widget (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect attribute), 454
 temp_font_path (in module kivymd.tools.release.update_icons), 499
 temp_path (in module kivymd.tools.release.update_icons), 499
 temp_preview_path (in module kivymd.tools.release.update_icons), 499
 temp_repo_path (in module kivymd.tools.release.update_icons), 499
 terminal_velocity (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect attribute), 458
 tertiary_font_style (kivymd.uix.list.list.BaseListItem attribute), 406
 tertiary_text (kivymd.uix.list.list.BaseListItem attribute), 405
 tertiary_text_color (kivymd.uix.list.list.BaseListItem attribute), 405
 tertiary_theme_text_color (kivymd.uix.list.list.BaseListItem attribute), 406
 text (kivymd.uix.banner.banner.MDBanner attribute), 195
 text (kivymd.uix.bottomnavigation.bottomnavigation.MDTabs attribute), 385
 text (kivymd.uix.button.button.BaseButton attribute), 109
 text (kivymd.uix.chip.chip.MDChip attribute), 415
 text (kivymd.uix.dialog.dialog.MDDialog attribute), 366
 text (kivymd.uix.dropdownitem.dropdownitem.MDDropDownItem attribute), 380
 text (kivymd.uix.label.label.MDLabel attribute), 257
 text (kivymd.uix.list.list.BaseListItem attribute), 405
 text (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 324
 text (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 321
 text (kivymd.uix.navigationrail.navigationrail.MDNavigationRailItem attribute), 164
 text (kivymd.uix.snackbar.snackbar.Snackbar attribute), 309
 text_button_cancel (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 241
 text_button_color (kivymd.uix.pickers.datepicker.datepicker.BaseDatePicker attribute), 227
 text_button_ok (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker attribute), 241
 text_color (kivymd.theming.ThemeManager attribute), 19
 text_color (kivymd.uix.button.button.BaseButton attribute), 110
 text_color (kivymd.uix.chip.chip.MDChip attribute), 415
 text_color (kivymd.uix.label.label.MDLabel attribute), 257
 text_color (kivymd.uix.list.list.BaseListItem attribute), 405
 text_color (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 325
 text_color (kivymd.uix.pickers.datepicker.datepicker.BaseDatePicker attribute), 227
 text_color_active (kivymd.uix.bottomnavigation.bottomnavigation.MDTabs attribute), 300
 text_color_active (kivymd.uix.tab.tab.MDTabs attribute), 300
 text_color_focus (kivymd.uix.textfield.textfield.MDTextField attribute), 343
 text_color_item_active (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 176
 text_color_item_normal (kivymd.uix.navigationrail.navigationrail.MDNavigationRail attribute), 175
 text_color_normal (kivymd.uix.bottomnavigation.bottomnavigation.MDTabs attribute), 386
 text_color_normal (kivymd.uix.tab.tab.MDTabs attribute), 300
 text_color_normal (kivymd.uix.textfield.textfield.MDTextField attribute), 343
 text_colors (in module kivymd.color_definitions), 29
 text_current_color (kivymd.uix.pickers.datepicker.datepicker.BaseDatePicker attribute), 228
 text_font_size (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 325
 text_font_style (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer attribute), 325

text_halign (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
text_right_color (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerItem attribute), 326
text_toolbar_color (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker attribute), 226
text_weekday_color (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker attribute), 234
ThemableBehavior (class in kivymd.theming), 24
theme_cls (kivymd.app.MDApp attribute), 26
theme_cls (kivymd.theming.ThemableBehavior attribute), 24
theme_colors (in module kivymd.color_definitions), 30
theme_font_styles (in module kivymd.font_definitions), 32
theme_icon_color (kivymd.uix.button.button.BaseButton attribute), 110
theme_style (kivymd.theming.ThemeManager attribute), 16
theme_text_color (kivymd.uix.button.button.BaseButton attribute), 110
theme_text_color (kivymd.uix.label.label.MDLabel attribute), 257
theme_text_color (kivymd.uix.list.list.BaseListItem attribute), 405
ThemeManager (class in kivymd.theming), 11
ThreeLineAvatarIconListItem (class in kivymd.uix.list.list), 408
ThreeLineAvatarListItem (class in kivymd.uix.list.list), 408
ThreeLineIconListItem (class in kivymd.uix.list.list), 408
ThreeLineListItem (class in kivymd.uix.list.list), 408
ThreeLineRightIconListItem (class in kivymd.uix.list.list), 408
thumb_color_active (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch attribute), 354
thumb_color_active (kivymd.uix.slider.slider.MDSlider attribute), 98
thumb_color_disabled (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch attribute), 355
thumb_color_disabled (kivymd.uix.slider.slider.MDSlider attribute), 98
thumb_color_inactive (kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch attribute), 354
thumb_color_inactive (kivymd.uix.slider.slider.MDSlider attribute), 98
time (kivymd.uix.pickers.timepicker.timepicker.MDTimePicker attribute), 247
title (kivymd.uix.backdrop.backdrop.MDBackdrop attribute), 327
title (kivymd.uix.dialog.dialog.MDDialog attribute), 224
title (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
title (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker attribute), 222
title (kivymd.uix.tab.tab.MDTabsBase attribute), 298
title (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 148
title_color (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
title_font_size (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
title_font_style (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
title_halign (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawerHeader attribute), 324
title_icon_mode (kivymd.uix.tab.tab.MDTabs attribute), 301
title_icon_mode (kivymd.uix.tab.tab.MDTabsBase attribute), 298
title_input (kivymd.uix.pickers.datepicker.datepicker.BaseDialogPicker attribute), 222
title_is_capital (kivymd.uix.tab.tab.MDTabsBase attribute), 299
title_position (kivymd.uix.taptargetview.MDTapTargetView attribute), 49
title_text (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
title_text_bold (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
title_text_color (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
title_text_size (kivymd.uix.taptargetview.MDTapTargetView attribute), 48
Toast (class in kivymd.toast.kivytoast.kivytoast), 482
toast() (in module kivymd.toast.androidtoast.androidtoast), 480
toast() (in module kivymd.toast.kivytoast.kivytoast), 482
toast() (kivymd.toast.kivytoast.kivytoast.Toast method), 482
toolbar_cls (kivymd.uix.sliverappbar.sliverappbar.MDSliverAppBar attribute), 129
tooltip_bg_color (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 190
tooltip_bg_color (kivymd.uix.tooltip.tooltip.MDTooltipViewClass attribute), 192
tooltip_display_delay (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 191
tooltip_font_style (kivymd.uix.tooltip.tooltip.MDTooltip attribute), 191

[tooltip_font_style](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 192
[tooltip_radius](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 191
[tooltip_radius](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 192
[tooltip_text](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 190
[tooltip_text](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 192
[tooltip_text_color](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 190
[tooltip_text_color](#) (*kivymd.uix.tooltip.tooltip.MDTooltipViewClass* attribute), 192
[TOUCH_TARGET_HEIGHT](#) (in module *kivymd.material_resources*), 478
[TouchBehavior](#) (class in *kivymd.uix.behaviors.touch_behavior*), 419
[track_color_active](#) (*kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch* attribute), 355
[track_color_active](#) (*kivymd.uix.slider.slider.MDSlider* attribute), 98
[track_color_disabled](#) (*kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch* attribute), 356
[track_color_disabled](#) (*kivymd.uix.slider.slider.MDSlider* attribute), 99
[track_color_inactive](#) (*kivymd.uix.selectioncontrol.selectioncontrol.MDSwitch* attribute), 356
[track_color_inactive](#) (*kivymd.uix.slider.slider.MDSlider* attribute), 99
[transformation_from_dialog_input_date\(\)](#) (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker* method), 237
[transformation_from_dialog_select_year\(\)](#) (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker* method), 236
[transformation_to_dialog_input_date\(\)](#) (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker* method), 237
[transformation_to_dialog_select_year\(\)](#) (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker* method), 237
[transition_duration](#) (*kivymd.uix.swiper.swiper.MDSwiper* attribute), 392
[transition_max](#) (*kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect* attribute), 454
[transition_min](#) (*kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect* attribute), 454
[Translation](#) (class in *kivymd.tools.patterns.MVC.libs.translation*), 497
[twist\(\)](#) (*kivymd.uix.behaviors.magic_behavior.MagicBehavior* method), 437
[TwoLineAvatarIconListItem](#) (class in *kivymd.uix.list.list*), 408
[TwoLineAvatarListItem](#) (class in *kivymd.uix.list.list*), 408
[TwoLineIconListItem](#) (class in *kivymd.uix.list.list*), 408
[TwoLineListItem](#) (class in *kivymd.uix.list.list*), 408
[TwoLineRightIconListItem](#) (class in *kivymd.uix.list.list*), 408
[type](#) (*kivymd.uix.banner.banner.MDBanner* attribute), 196
[type](#) (*kivymd.uix.button.button.MDFloatingActionButton* attribute), 112
[type](#) (*kivymd.uix.dialog.dialog.MDDialog* attribute), 371
[type](#) (*kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer* attribute), 327
[type](#) (*kivymd.uix.navigationrail.navigationrail.MDNavigationRail* attribute), 173
[type](#) (*kivymd.uix.progressbar.progressbar.MDProgressBar* attribute), 188
[type](#) (*kivymd.uix.toolbar.toolbar.MDTopAppBar* attribute), 150
[type_color](#) (*kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker* attribute), 240
[type_height](#) (*kivymd.uix.toolbar.toolbar.MDTopAppBar* attribute), 153
[type_swipe](#) (*kivymd.uix.card.card.MDCardSwipe* attribute), 209

U

[uix_path](#) (in module *kivymd*), 477
[underline_color](#) (*kivymd.uix.tab.tab.MDTabs* attribute), 300
[unfocus_color](#) (*kivymd.uix.behaviors.focus_behavior.FocusBehavior* attribute), 453
[unload_app_dependencies\(\)](#) (*kivymd.tools.hotreload.app.MDApp* method), 487
[unselected_all\(\)](#) (*kivymd.uix.selection.selection.MDSelectionList* method), 217
[unselected_color](#) (*kivymd.uix.selectioncontrol.selectioncontrol.MDCheckbox* attribute), 351
[unzip_archive\(\)](#) (in module *kivymd.tools.release.update_icons*), 500
[update\(\)](#) (*kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect* method), 455
[update_action_bar\(\)](#) (*kivymd.uix.toolbar.toolbar.MDTopAppBar* method), 154

`update_action_bar_text_colors()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_anchor_title()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_background_origin()` (kivymd.uix.behaviors.backgroundcolor_behavior.BackgroundColorBehavior method), 425
`update_bar_height()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_calendar()` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
`update_calendar_for_date_range()` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
`update_canvas()` (kivymd.effects.fadingedge.fadingedge.FadingEdgeEffect method), 456
`update_color()` (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox method), 352
`update_color_slider_item_bottom_navigation()` (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
`update_color_type_buttons()` (kivymd.uix.pickers.colorpicker.colorpicker.MDColorPicker method), 241
`update_floating_radius()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_font_style()` (kivymd.uix.label.label.MDLabel method), 257
`update_fps()` (kivymd.utils.fpsmonitor.FpsMonitor method), 508
`update_group_property()` (kivymd.uix.behaviors.elevation.CommonElevationBehavior method), 449
`update_height()` (kivymd.uix.dialog.dialog.MDDialog method), 373
`update_icon()` (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox method), 352
`update_icon_color()` (kivymd.uix.tab.tab.MDTabs method), 301
`update_icons()` (in module kivymd.tools.release.update_icons), 500
`update_init_py()` (in module kivymd.tools.release.make_release), 499
`update_items()` (kivymd.uix.dialog.dialog.MDDialog method), 373
`update_label_text()` (kivymd.uix.tab.tab.MDTabsBase method), 299
`update_md_bg_color()` (kivymd.uix.card.card.MDCard method), 208
`update_md_bg_color()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_overflow_menu_items()` (kivymd.uix.toolbar.toolbar.MDTopAppBar method), 154
`update_pos()` (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 320
`update_primary_color()` (kivymd.uix.selectioncontrol.selectioncontrol.MDCheckBox method), 351
`update_readme()` (in module kivymd.tools.release.make_release), 499
`update_row()` (kivymd.uix.datatables.datatables.MDDDataTable method), 94
`update_scroll_offset_data()` (kivymd.uix.datatables.datatables.MDDDataTable method), 92
`update_scroll_rectangle()` (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 320
`update_segment_panel_width()` (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl method), 361
`update_separator_color()` (kivymd.uix.segmentedcontrol.segmentedcontrol.MDSegmentedControl method), 361
`update_status()` (kivymd.uix.navigationdrawer.navigationdrawer.MDNavigationDrawer method), 331
`update_text_full_date()` (kivymd.uix.pickers.datepicker.datepicker.MDDatePicker method), 237
`update_velocity()` (kivymd.effects.roulettescroll.roulettescroll.RouletteScrollEffect method), 458
`update_velocity()` (kivymd.effects.stiffscroll.stiffscroll.StiffScrollEffect method), 454
`update_width()` (kivymd.uix.dialog.dialog.MDDialog method), 373
`updated_interval` (kivymd.utils.fpsmonitor.FpsMonitor attribute), 508
`upload_file()` (kivymd.tools.patterns.MVC.Model.database_restdb.DatabaseRestDb method), 497
`url` (in module kivymd.tools.release.update_icons), 499
`use_access` (kivymd.uix.filemanager.filemanager.MDFileManager attribute), 377
`use_overflow` (kivymd.uix.toolbar.toolbar.MDTopAppBar attribute), 151
`use_pagination` (kivymd.uix.datatables.datatables.MDDDataTable attribute), 88
`use_text` (kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigationBar attribute), 386

V

`valign` (*kivymd.uix.button.button.BaseButton* attribute),
109

`value_transparent` (*kivymd.uix.bottomsheet.bottomsheet.MDBottomSheet*
attribute), 123

`ver_growth` (*kivymd.uix.menu.menu.MDDropdownMenu*
attribute), 274

`vertical_pad` (*kivymd.uix.banner.banner.MDBanner*
attribute), 195

W

`widget` (*kivymd.uix.taptargetview.MDTapTargetView* at-
tribute), 46

`widget_index` (*kivymd.uix.bottomnavigation.bottomnavigation.MDBottomNavigation*
attribute), 387

`widget_position` (*kivymd.uix.taptargetview.MDTapTargetView*
attribute), 49

`widget_style` (*kivymd.theming.ThemableBehavior* at-
tribute), 24

`width_mult` (*kivymd.uix.menu.menu.MDDropdownMenu*
attribute), 269

`width_mult` (*kivymd.uix.swiper.swiper.MDSwiper*
attribute), 393

`width_offset` (*kivymd.uix.dialog.dialog.MDDialog* at-
tribute), 371

`WindowController` (class in
kivymd.uix.controllers.windowcontroller),
417

`wobble()` (*kivymd.uix.behaviors.magic_behavior.MagicBehavior*
method), 437

Y

`year` (*kivymd.uix.pickers.datepicker.datepicker.MDDatePicker*
attribute), 235