KivyMD

Release 2.0.1.dev0

Andrés Rodríguez, Ivanov Yuri, Artem Bulgakov and KivyMD cont

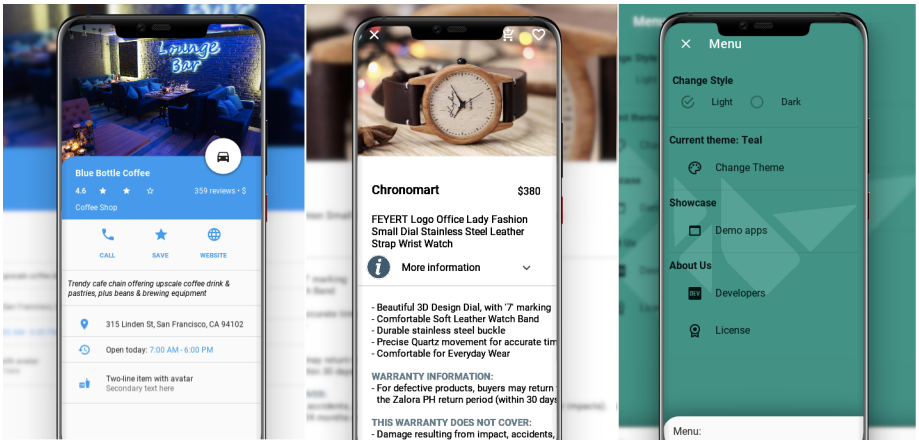
Apr 14, 2024

CONTENTS

CHAPTER

ONE

KIVYMD



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

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.

CHAPTER

TWO

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.uix.screen import MDScreen
from kivymd.uix.button import MDButton, MDButtonText


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

        return (
            MDScreen(
                MDButton(
                    MDButtonText(
                        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, Dynamic color

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.

API - kivymd.theming

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

Dynamic color class.

New in version 2.0.0.

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.

See `kivy.utils.hex_colormap` keys for available values.

To change the color scheme of an application:

Imperative python style with KV

```
from kivy.lang import Builder

from kivymd.app import MDApp

```

(continues on next page)

(continued from previous page)

```
KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        style: "elevated"
        pos_hint: {"center_x": .5, "center_y": .5}

        MDButtonIcon:
            icon: "plus"

        MDButtonText:
            text: "Button"
'''


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


Example().run()
```

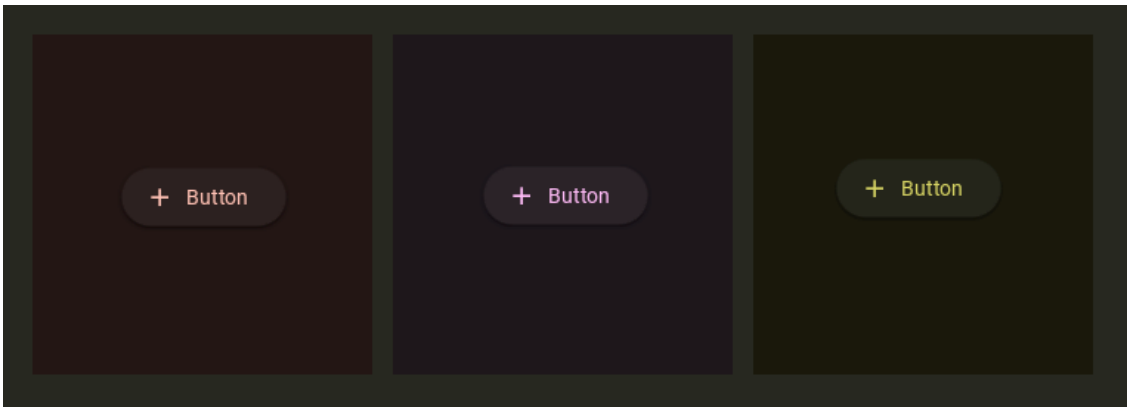
Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonIcon, MDButtonText
from kivymd.uix.screen import MDScreen


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

        return (
            MDScreen(
                MDButton(
                    MDButtonIcon(
                        icon="plus",
                    ),
                    MDButtonText(
                        text="Button",
                    ),
                    style="elevated",
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                ),
                md_bg_color=self.theme_cls.backgroundColor,
            )
        )
```

(continues on next page)

(continued from previous page)

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



primary_palette is an OptionProperty and defaults to *None*.

dynamic_color_quality

The quality of the generated color scheme from the system wallpaper. It is equal to or higher than *1*, with *1* representing the maximum quality.

> **Warning:** Remember that by increasing the quality value, you also increase the generation time of the color scheme.

dynamic_color_quality is an NumericProperty and defaults to *10* if platform is not Android else *1*.

dynamic_color

Enables or disables dynamic color.

New in version 2.0.0.

See also:

Material Design spec, Dynamic color

To build the color scheme of your application from user wallpapers, you must enable the READ_EXTERNAL_STORAGE permission if your android version is below 8.1:

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

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: app.theme_cls.surfaceColor

    MDButton:
        style: "elevated"
        pos_hint: {"center_x": .5, "center_y": .5}
```

(continues on next page)

(continued from previous page)

```
        MDButtonIcon:
            icon: "plus"

        MDButtonText:
            text: "Elevated"
'''


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

    def on_resume(self, *args):
        '''Updating the color scheme when the application resumes.'''

        self.theme_cls.set_colors()

    def set_dynamic_color(self, *args) -> None:
        '''
        When sets the `dynamic_color` value, the self method will be
        `called.theme_cls.set_colors()` which will generate a color
        scheme from a custom wallpaper if `dynamic_color` is `True`.
        '''

        self.theme_cls.dynamic_color = True

    def on_start(self) -> None:
        '''
        It is fired at the start of the application and requests the
        necessary permissions.
        '''

        def callback(permission, results):
            if all([res for res in results]):
                Clock.schedule_once(self.set_dynamic_color)

        if platform == "android":
            from android.permissions import Permission, request_permissions

            permissions = [Permission.READ_EXTERNAL_STORAGE]
            request_permissions(permissions, callback)


Example().run()
```

dynamic_color is an BooleanProperty and defaults to *False*.

dynamic_scheme_name

Name of the dynamic scheme. Availabe schemes *TONAL_SPOT*, *SPRITZ VIBRANT*, *EXPRESSIVE*, *FRUIT_SALAD*, *RAINBOW*, *MONOCHROME*, *FIDELITY* and *CONTENT*.

dynamic_scheme_name is an OptionProperty and defaults to *'TONAL_SPOT'*.

dynamic_scheme_contrast

The contrast of the generated color scheme.

dynamic_scheme_contrast is an NumericProperty and defaults to *0.0*.

path_to_wallpaper

The path to the image to set the color scheme. You can use this option if you want to use dynamic color on platforms other than the Android platform.

New in version 2.0.0.

path_to_wallpaper is an StringProperty and defaults to ‘’.

theme_style_switch_animation

Animate app colors when switching app color scheme (‘Dark/light’).

New in version 1.1.0.

Declarative KV style

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDCard:
        orientation: "vertical"
        padding: 0, 0, 0 , "36dp"
        size_hint: .5, .5
        style: "elevated"
        pos_hint: {"center_x": .5, "center_y": .5}

        MDLabel:
            text: "Theme style - {}".format(app.theme_cls.theme_style)
            halign: "center"
            valign: "center"
            bold: True
            font_style: "Display"
            role: "small"

        MDButton:
            on_release: app.switch_theme_style()
            pos_hint: {"center_x": .5}

            MDButtonText:
                text: "Set theme"
'''


class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style_switch_animation = True
        self.theme_cls.theme_style = "Dark"
```

(continues on next page)

(continued from previous page)

```
        self.theme_cls.primary_palette = "Orange"
        return Builder.load_string(KV)

    def switch_theme_style(self):
        self.theme_cls.primary_palette = (
            "Orange" if self.theme_cls.primary_palette == "Red" else "Red"
        )
        self.theme_cls.theme_style = (
            "Dark" if self.theme_cls.theme_style == "Light" else "Light"
        )


Example().run()
```

Declarative python style

```
from kivy.clock import Clock

from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.card import MDCard
from kivymd.uix.label import MDLabel
from kivymd.uix.screen import MDScreen


class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style_switch_animation = True
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDScreen(
                MDCard(
                    MDLabel(
                        id="label",
                        text="Theme style - {}".format(
                            self.theme_cls.theme_style),
                        halign="center",
                        valign="center",
                        bold=True,
                        font_style="Display",
                        role="small",
                    ),
                    MDButton(
                        MDButtonText(
                            text="Set theme",
                        ),
                        on_release=self.switch_theme_style,
                        pos_hint={"center_x": 0.5},
                    ),
                    id="card",
                    orientation="vertical",
```

(continues on next page)

(continued from previous page)

```
                    padding=(0, 0, 0, "36dp"),
                    size_hint=(0.5, 0.5),
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                    style="elevated",
                )
            )
        )

    def on_start(self):
        def on_start(*args):
            self.root.md_bg_color = self.theme_cls.backgroundColor

        Clock.schedule_once(on_start)

    def switch_theme_style(self, *args):
        self.theme_cls.primary_palette = (
            "Orange" if self.theme_cls.primary_palette == "Red" else "Red"
        )
        self.theme_cls.theme_style = (
            "Dark" if self.theme_cls.theme_style == "Light" else "Light"
        )
        self.root.get_ids().label.text = (
            "Theme style - {}".format(self.theme_cls.theme_style)
        )


Example().run()
```

theme_style_switch_animation is an BooleanProperty and defaults to *True*.

theme_style_switch_animation_duration

Duration of the animation of switching the color scheme of the application ("Dark/light").

New in version 1.1.0.

```
class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style_switch_animation = True
        self.theme_cls.theme_style_switch_animation_duration = 0.8
```

theme_style_switch_animation_duration is an NumericProperty and defaults to *0.2*.

theme_style

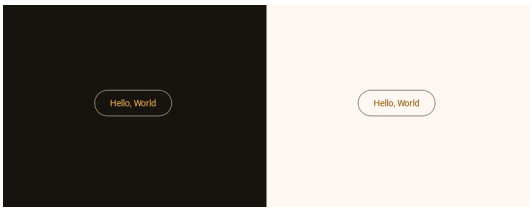
App theme style.

```
from kivy.clock import Clock

from kivymd.app import MDApp
from kivymd.uix.screen import MDScreen
from kivymd.uix.button import MDButton, MDButtonText
```

(continues on next page)

(continued from previous page)

```
class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Light"  # "Dark"
        return MDScreen(
            MDButton(
                MDButtonText(
                    text="Hello, World",
                ),
                style="outlined",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            )
        )

    def on_start(self):
        def on_start(*args):
            self.root.md_bg_color = self.theme_cls.backgroundColor

        Clock.schedule_once(on_start)


Example().run()
```



theme_style is an OptionProperty and defaults to *'Light'*.

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.

device_orientation

Device orientation.

device_orientation is an StringProperty and defaults to ''.

font_styles

Data of default font styles.

Add custom font

Declarative style with KV

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

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

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


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

        LabelBase.register(
            name="nasalization",
            fn_regular="nasalization.ttf",
        )

        self.theme_cls.font_styles["nasalization"] = {
            "large": {
                "line-height": 1.64,
                "font-name": "nasalization",
                "font-size": sp(57),
            },
            "medium": {
                "line-height": 1.52,
                "font-name": "nasalization",
                "font-size": sp(45),
            },
            "small": {
                "line-height": 1.44,
                "font-name": "nasalization",
                "font-size": sp(36),
            },
        }

        return Builder.load_string(KV)


Example().run()
```

Declarative python style

```
from kivy.core.text import LabelBase
from kivy.metrics import sp

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


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

        LabelBase.register(
            name="nasalization",
            fn_regular="/Users/urijivanov/Projects/Dev/MyGithub/Articles/
↪StarTest/data/font/nasalization-rg.ttf",
        )

        self.theme_cls.font_styles["nasalization"] = {
            "large": {
                "line-height": 1.64,
                "font-name": "nasalization",
                "font-size": sp(57),
            },
            "medium": {
                "line-height": 1.52,
                "font-name": "nasalization",
                "font-size": sp(45),
            },
            "small": {
                "line-height": 1.44,
                "font-name": "nasalization",
                "font-size": sp(36),
            },
        }

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


Example().run()
```

font_styles is an DictProperty.

on_colors

A Helper function called when colors are changed.

Attr

on_colors defaults to *None*.

set_colors(**args*) → None

Fired methods for setting a new color scheme.

update_theme_colors(**args*) → None

Fired when the *theme_style* value changes.

on_dynamic_scheme_name(**args*)

on_dynamic_scheme_contrast(**args*)

on_path_to_wallpaper(**args*)

switch_theme() → None

Switches the theme from light to dark.

sync_theme_styles(**args*) → None

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.

theme_line_color

Line color scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_line_color is an OptionProperty and defaults to *'Primary'*.

theme_bg_color

Background color scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_bg_color is an OptionProperty and defaults to *'Primary'*.

theme_shadow_color

Elevation color scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_shadow_color is an OptionProperty and defaults to *'Primary'*.

theme_shadow_offset

Elevation offset scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_shadow_offset is an OptionProperty and defaults to *'Primary'*.

theme_elevation_level

Elevation level scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_elevation_level is an OptionProperty and defaults to *'Primary'*.

theme_font_size

Font size scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_font_size is an OptionProperty and defaults to *'Primary'*.

theme_width

Widget width scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_width is an OptionProperty and defaults to *'Primary'*.

theme_height

Widget width scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_height is an OptionProperty and defaults to *'Primary'*.

theme_line_height

Line height scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_line_height is an OptionProperty and defaults to *'Primary'*.

theme_font_name

Font name scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_font_name is an OptionProperty and defaults to *'Primary'*.

theme_shadow_softness

Elevation softness scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_shadow_softness is an OptionProperty and defaults to *'Primary'*.

theme_focus_color

Focus color scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_focus_color is an OptionProperty and defaults to *'Primary'*.

theme_divider_color

Divider color scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_divider_color is an OptionProperty and defaults to *'Primary'*.

theme_text_color

Label color scheme name.

Available options are: *'Primary'*, *'Secondary'*, *'Hint'*, *'Error'*, *'Custom'*.

theme_text_color is an OptionProperty and defaults to *'Primary'*.

theme_icon_color

Label color scheme name.

Available options are: *'Primary'*, *'Secondary'*, *'Hint'*, *'Error'*, *'Custom'*.

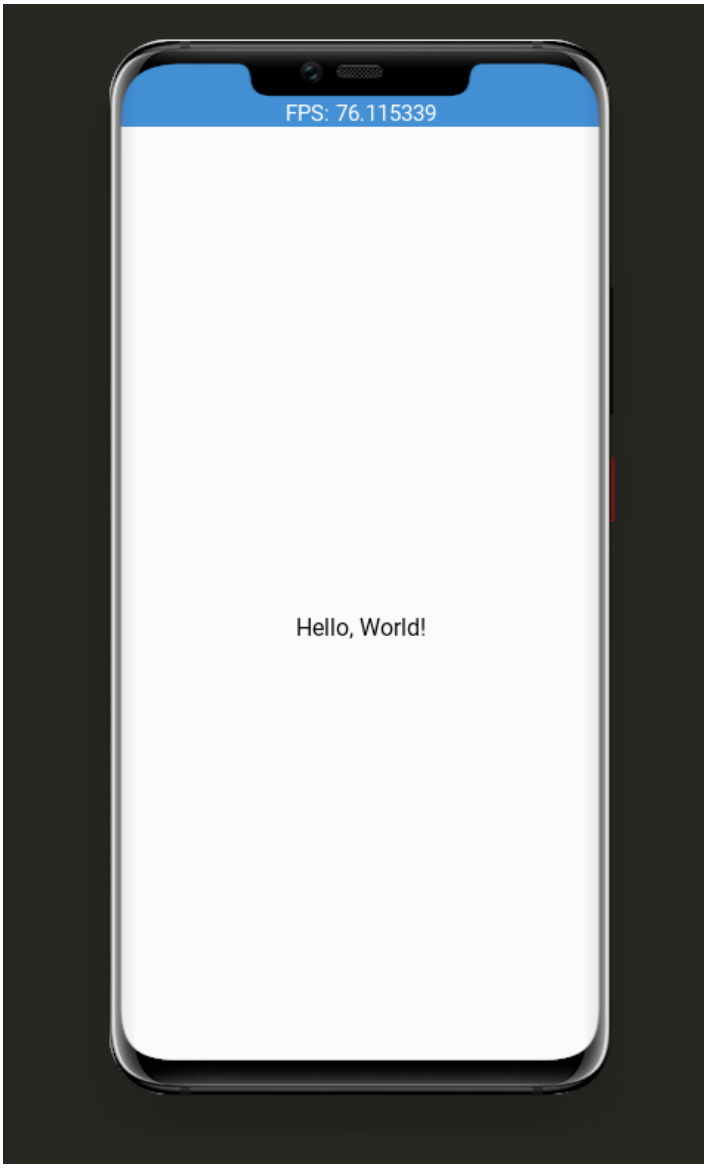
theme_icon_color is an OptionProperty and defaults to *'Primary'*.

remove_widget(*widget*) → None

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 *FP* value in your application:

```
KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    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()
```



Note: Note that if you override the built-in on_start method, you will definitely need to call the super method:

```
class MainApp(MDApp):
    def build(self):
        [...]

    def on_start(self):
        [...]
```

API - kivymd.app

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

Application class, see App class documentation for more information.

icon

See icon attribute for more information.

New in version 1.1.0.

icon is an StringProperty adn default to *kivymd/images/logo/kivymd-icon-512.png*.

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"
```

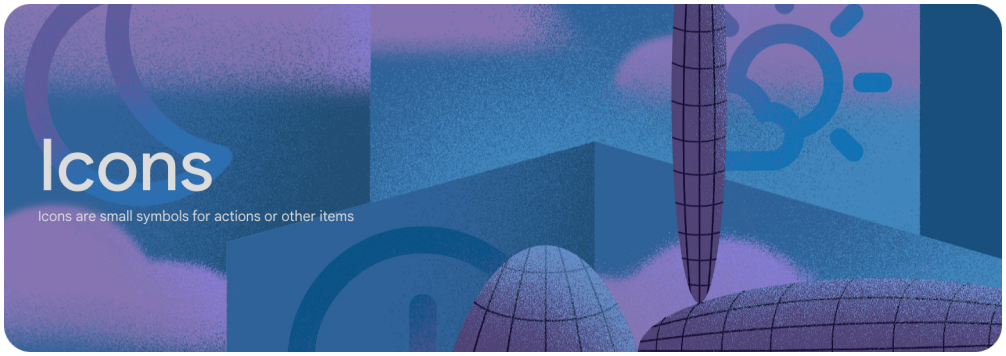
theme_cls is an ObjectProperty.

load_all_kv_files(*path_to_directory: str*) → None

Recursively loads KV files from the selected directory.

New in version 1.0.0.

2.2.3 Icon Definitions



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

Version 7.4.47

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

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

from kivymd.icon_definitions import md_icons
from kivymd.uix.screen import MDScreen
from kivymd.app import MDApp
from kivymd.uix.list import MDListItem

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


<IconItem>

    MDListItemLeadingIcon:
        icon: root.icon

    MDListItemSupportingText:
        text: root.text


<PreviousMDIcons>
    md_bg_color: self.theme_cls.backgroundColor

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

        MDBoxLayout:
            adaptive_height: True

            MDIconButton:
                icon: 'magnify'
                pos_hint: {'center_y': .5}

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

        RecycleView:
            id: rv
            key_viewclass: 'viewclass'
```

(continues on next page)

(continued from previous page)

```
            key_size: 'height'

            RecycleBoxLayout:
                padding: dp(10), dp(10), 0, dp(10)
                default_size: None, dp(48)
                default_size_hint: 1, None
                size_hint_y: None
                height: self.minimum_height
                orientation: 'vertical'
'''
)


class IconItem(MDListItem):
    icon = StringProperty()
    text = StringProperty()


class PreviousMDIcons(MDScreen):
    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": "IconItem",
                    "icon": name_icon,
                    "text": name_icon,
                    "callback": lambda x: x,
                }
            )

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


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

    def build(self):
        return self.screen

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

(continues on next page)

(continued from previous page)

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

API - kivymd.icon_definitions

kivymd.icon_definitions.**md_icons**

class kivymd.icon_definitions.**IconItem**(**args*, ***kwargs*)

Implements a list item.

For more information, see in the `BaseListItem` and `BoxLayout` classes documentation.

icon

text

2.2.4 Font definitions

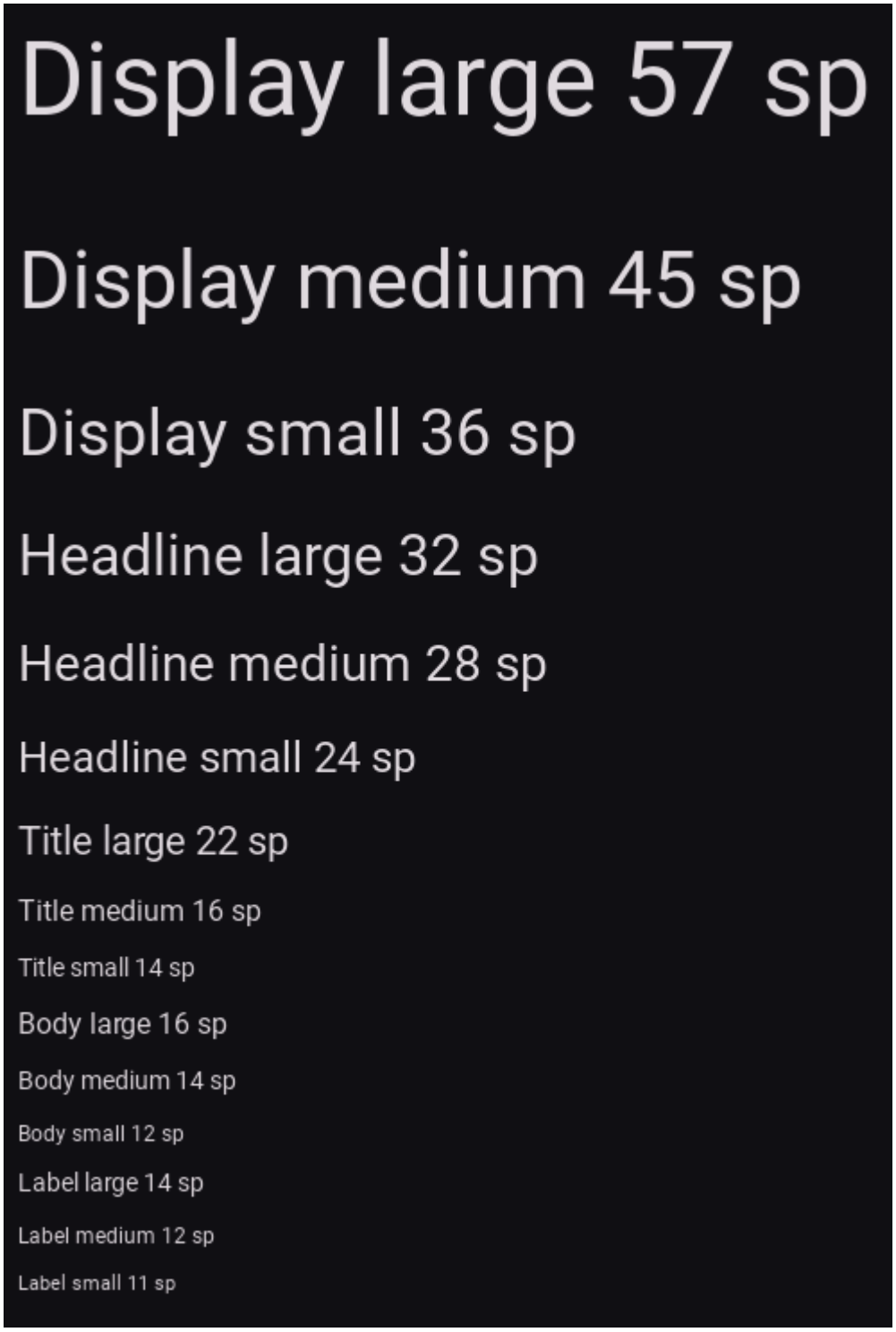
See also:

Material Design spec, The type system

API - kivymd.font_definitions

kivymd.font_definitions.**fonts**

kivymd.font_definitions.**theme_font_styles**


Display large 57 sp
Display medium 45 sp
Display small 36 sp
Headline large 32 sp
Headline medium 28 sp
Headline small 24 sp
Title large 22 sp
Title medium 16 sp
Title small 14 sp
Body large 16 sp
Body medium 14 sp
Body small 12 sp
Label large 14 sp
Label medium 12 sp
Label small 11 sp

2.3 Components

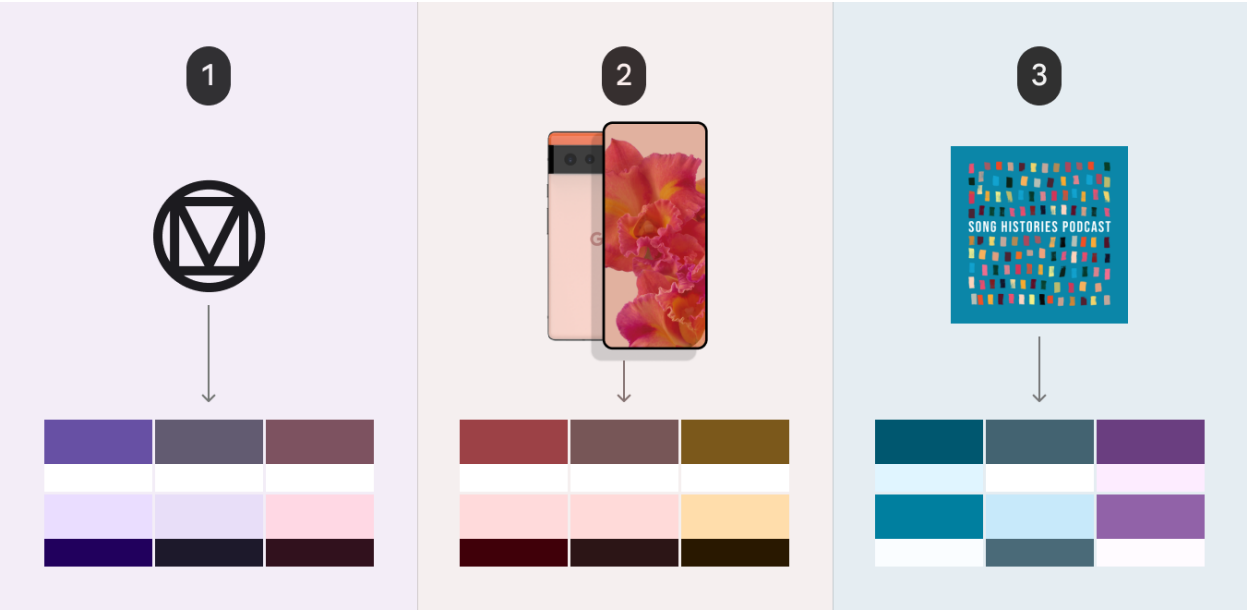
2.3.1 Dynamic color

See also:

Material Design spec, Dynamic color

Dynamic color can create accessible UI color schemes based on content or user settings

Dynamic color experiences are built with M3 color schemes. Beginning with Android 12, users can generate individualized schemes through wallpaper selection and other customization settings. With M3 as a foundation, user-generated colors can coexist with app colors, putting a range of customizable visual experiences in the hands of users.



1. Baseline scheme
2. Colors extracted from a wallpaper
3. Colors extracted from content

Example of dynamic color from the list of standard color schemes

```
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.properties import StringProperty, ColorProperty
from kivy.uix.boxlayout import BoxLayout
from kivy.utils import hex_colormap

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


KV = '''
<ColorCard>
    orientation: "vertical"

    MDLabel:
        text: root.text
        color: "grey"
        adaptive_height: True

    MDCard:
        theme_bg_color: "Custom"
        md_bg_color: root.bg_color


MDScreen:
    md_bg_color: app.theme_cls.backgroundColor

    MDIconButton:
        on_release: app.open_menu(self)
        pos_hint: {"top": .98}
        x: "12dp"
        icon: "menu"

    MDRecycleView:
        id: card_list
        viewclass: "ColorCard"
        bar_width: 0
        size_hint_y: None
        height: root.height - dp(68)

        RecycleGridLayout:
            cols: 3
            spacing: "16dp"
            padding: "16dp"
            default_size: None, dp(56)
```

(continues on next page)

(continued from previous page)

```
            default_size_hint: 1, None
            size_hint_y: None
            height: self.minimum_height
'''


class ColorCard(BoxLayout):
    text = StringProperty()
    bg_color = ColorProperty()


class Example(MDApp):
    menu: MDDropdownMenu = None

    def build(self):
        self.theme_cls.dynamic_color = True
        return Builder.load_string(KV)

    def get_instance_from_menu(self, name_item):
        index = 0
        rv = self.menu.ids.md_menu
        opts = rv.layout_manager.view_opts
        datas = rv.data[0]

        for data in rv.data:
            if data["text"] == name_item:
                index = rv.data.index(data)
                break

        instance = rv.view_adapter.get_view(
            index, datas, opts[index]["viewclass"]
        )

        return instance

    def open_menu(self, menu_button):
        menu_items = []
        for item, method in {
            "Set palette": lambda: self.set_palette(),
            "Switch theme style": lambda: self.theme_switch(),
        }.items():
            menu_items.append({"text": item, "on_release": method})
        self.menu = MDDropdownMenu(
            caller=menu_button,
            items=menu_items,
        )
        self.menu.open()

    def set_palette(self):
        instance_from_menu = self.get_instance_from_menu("Set palette")
        available_palettes = [
            name_color.capitalize() for name_color in hex_colormap.keys()
```

(continues on next page)

(continued from previous page)

```
        ]

        menu_items = []
        for name_palette in available_palettes:
            menu_items.append(
                {
                    "text": name_palette,
                    "on_release": lambda x=name_palette: self.switch_palette(x),
                }
            )
        MDDropdownMenu(
            caller=instance_from_menu,
            items=menu_items,
        ).open()

    def switch_palette(self, selected_palette):
        self.theme_cls.primary_palette = selected_palette
        Clock.schedule_once(self.generate_cards, 0.5)

    def theme_switch(self) -> None:
        self.theme_cls.switch_theme()
        Clock.schedule_once(self.generate_cards, 0.5)

    def generate_cards(self, *args):
        self.root.ids.card_list.data = []
        for color in self.theme_cls.schemes_name_colors:
            value = f"{color}Color"
            self.root.ids.card_list.data.append(
                {
                    "bg_color": getattr(self.theme_cls, value),
                    "text": value,
                }
            )

    def on_start(self):
        Clock.schedule_once(self.generate_cards)


Example().run()
```

Example of a dynamic color from an image

See also:

kivymd.theming.ThemeManager.path_to_wallpaper

```
import os

from kivy.clock import Clock
from kivy.core.window import Window
```

(continues on next page)

(continued from previous page)

```
from kivy.core.window.window_sdl2 import WindowSDL
from kivy.lang import Builder
from kivy.properties import StringProperty, ColorProperty

from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.app import MDApp


KV = '''
<ColorCard>
    orientation: "vertical"

    MDLabel:
        text: root.text
        color: "grey"
        adaptive_height: True

    MDCard:
        theme_bg_color: "Custom"
        md_bg_color: root.bg_color


MDScreen:
    md_bg_color: app.theme_cls.backgroundColor

    MDRecycleView:
        id: card_list
        viewclass: "ColorCard"
        bar_width: 0

        RecycleGridLayout:
            cols: 3
            spacing: "16dp"
            padding: "16dp"
            default_size: None, dp(56)
            default_size_hint: 1, None
            size_hint_y: None
            height: self.minimum_height
'''


class ColorCard(MDBoxLayout):
    text = StringProperty()
    bg_color = ColorProperty()


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        Window.bind(on_dropfile=self.on_drop_file)

    def on_drop_file(self, sdl: WindowSDL, path_to_file: str) -> None:
```

(continues on next page)

(continued from previous page)

```
        ext = os.path.splitext(path_to_file)[1]
        if isinstance(path_to_file, bytes):
            path_to_file = path_to_file.decode()
        if isinstance(ext, bytes):
            ext = ext.decode()
        if ext in [".png", ".jpg"]:
            self.theme_cls.path_to_wallpaper = path_to_file
            Clock.schedule_once(self.generate_cards, 0.5)

    def build(self):
        self.theme_cls.dynamic_color = True
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

    def theme_switch(self) -> None:
        self.theme_cls.switch_theme()
        Clock.schedule_once(self.generate_cards, 0.5)

    def generate_cards(self, *args):
        self.root.ids.card_list.data = []
        for color in self.theme_cls.schemes_name_colors:
            value = f"{color}Color"
            self.root.ids.card_list.data.append(
                {
                    "bg_color": getattr(self.theme_cls, value),
                    "text": value,
                }
            )

    def on_start(self):
        Clock.schedule_once(self.generate_cards)


Example().run()
```

API - kivymd.dynamic_color

class kivymd.dynamic_color.**DynamicColor**

Dynamic color class.

New in version 2.0.0.

primaryColor

Primary color.

primaryColor is an ColorProperty and defaults to *None*.

primaryContainerColor

Primary container color.

primaryContainerColor is an ColorProperty and defaults to *None*.

onPrimaryColor

On primary color.

onPrimaryColor is an ColorProperty and defaults to *None*.

onPrimaryContainerColor

On primary container color.

onPrimaryContainerColor is an ColorProperty and defaults to *None*.

secondaryColor

Secondary color.

secondaryColor is an ColorProperty and defaults to *None*.

secondaryContainerColor

Secondary container color.

secondaryContainerColor is an ColorProperty and defaults to *None*.

onSecondaryColor

On secondary color.

onSecondaryColor is an ColorProperty and defaults to *None*.

onSecondaryContainerColor

On secondary container color.

onSecondaryContainerColor is an ColorProperty and defaults to *None*.

tertiaryColor

Tertiary color.

tertiaryColor is an ColorProperty and defaults to *None*.

tertiaryContainerColor

Tertiary container color.

tertiaryContainerColor is an ColorProperty and defaults to *None*.

onTertiaryColor

On tertiary color.

onTertiaryColor is an ColorProperty and defaults to *None*.

onTertiaryContainerColor

On tertiary container color.

onTertiaryContainerColor is an ColorProperty and defaults to *None*.

surfaceColor

Surface color.

surfaceColor is an ColorProperty and defaults to *None*.

surfaceDimColor

Surface dim color.

surfaceDimColor is an ColorProperty and defaults to *None*.

surfaceBrightColor

Surface bright color.

surfaceBrightColor is an ColorProperty and defaults to *None*.

surfaceContainerLowestColor

Surface container lowest color.

surfaceContainerLowestColor is an ColorProperty and defaults to *None*.

surfaceContainerLowColor

Surface container low color.

surfaceContainerLowColor is an ColorProperty and defaults to *None*.

surfaceContainerColor

Surface container color.

surfaceContainerColor is an ColorProperty and defaults to *None*.

surfaceContainerHighColor

Surface container high color.

surfaceContainerHighColor is an ColorProperty and defaults to *None*.

surfaceContainerHighestColor

Surface container highest color.

surfaceContainerHighestColor is an ColorProperty and defaults to *None*.

surfaceVariantColor

Surface variant color.

surfaceVariantColor is an ColorProperty and defaults to *None*.

surfaceTintColor

Surface tint color.

surfaceTintColor is an ColorProperty and defaults to *None*.

onSurfaceColor

On surface color.

onSurfaceColor is an ColorProperty and defaults to *None*.

onSurfaceLightColor

On surface light color.

onSurfaceLightColor is an ColorProperty and defaults to *None*.

onSurfaceVariantColor

On surface variant color.

onSurfaceVariantColor is an ColorProperty and defaults to *None*.

inverseSurfaceColor

Inverse surface color.

inverseSurfaceColor is an ColorProperty and defaults to *None*.

inverseOnSurfaceColor

Inverse on surface color.

inverseOnSurfaceColor is an ColorProperty and defaults to *None*.

inversePrimaryColor

Inverse primary color.

inversePrimaryColor is an ColorProperty and defaults to *None*.

backgroundColor

Background color.

backgroundColor is an ColorProperty and defaults to *None*.

onBackgroundColor

On background color.

onBackgroundColor is an ColorProperty and defaults to *None*.

errorColor

Error color.

errorColor is an ColorProperty and defaults to *None*.

errorContainerColor

Error container color.

errorContainerColor is an ColorProperty and defaults to *None*.

onErrorColor

On error color.

onErrorColor is an ColorProperty and defaults to *None*.

onErrorContainerColor

On error container color.

onErrorContainerColor is an ColorProperty and defaults to *None*.

outlineColor

Outline color.

outlineColor is an ColorProperty and defaults to *None*.

outlineVariantColor

Outline variant color.

outlineVariantColor is an ColorProperty and defaults to *None*.

shadowColor

Shadow color.

shadowColor is an ColorProperty and defaults to *None*.

scrimColor

Scrim color.

scrimColor is an ColorProperty and defaults to *None*.

disabledTextColor

Disabled text color.

disabledTextColor is an ColorProperty and defaults to *None*.

transparentColor

Transparent color.

transparentColor is an ColorProperty and defaults to *[0, 0, 0, 0]*.

rippleColor

Ripple color.

rippleColor is an ColorProperty and defaults to *'#BDBDBD'*.

2.3.2 FloatLayout

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

FloatLayout

```
FloatLayout:
    canvas:
        Color:
            rgba: app.theme_cls.primaryColor
        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.primaryColor
```

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 *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and FloatLayout and *MDAdaptiveWidget* classes documentation.

2.3.3 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDGridLayout

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

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*)

Grid layout class.

For more information see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and GridLayout and *MDAdaptiveWidget* classes documentation.

2.3.4 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDRecycleView

```
MDRecycleView:
    md_bg_color: app.theme_cls.primaryColor
```

API - kivymd.uix.recycleview

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

Recycle view class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and RecycleView and *MDAdaptiveWidget* classes documentation.

2.3.5 ScrollView

New in version 1.0.0.

ScrollView class equivalent. It implements Material Design's overscorll effect and simplifies working with some widget properties. For example:

ScrollView

```
ScrollView:

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

MDScrollView

```
MDScrollView:
    md_bg_color: app.theme_cls.primaryColor
```

API - kivymd.uix.scrollview

class kivymd.uix.scrollview.**StretchOverScrollStencil**(**arg*, ***kwargs*)

Stretches the view on overscroll and absorbs velocity at start and end to convert to stretch. .. note:: This effect only works with *kivymd.uix.scrollview.MDScrollView*. If you need any documentation please look at dampedscrolleffect.

minimum_absorbed_velocity = 0

maximum_velocity = 10000

stretch_intensity = 0.016

exponential_scalar

scroll_friction = 0.015

approx_normailzer = 200000.0

duration_normailzer = 10

scroll_view

scroll_scale

scale_axis = 'y'

last_touch_pos

clamp(*value*, *min_val=0*, *max_val=0*)

is_top_or_bottom()

on_value(*stencil*, *scroll_distance*)

get_hw()

set_scale_origin()

absorb_impact()

get_component(*pos*)

convert_overscroll(*touch*)

reset_scale(**arg*)

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

An approximate implementation to Material Design's overscorll effect.

For more information, see in the *DeclarativeBehavior* and *BackgroundColorBehavior* and ScrollView classes documentation.

on_touch_down(*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(*touch*)

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

See *on_touch_down()* for more information.

on_touch_up(*touch*)

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

See *on_touch_down()* for more information.

2.3.6 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDStackLayout

```
MDStackLayout:
    adaptive_height: True
    md_bg_color: app.theme_cls.primaryColor
```

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 *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and StackLayout and *MDAdaptiveWidget* classes documentation.

2.3.7 RelativeLayout

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

RelativeLayout

```
RelativeLayout:
    canvas:
        Color:
            rgba: app.theme_cls.primaryColor
        RoundedRectangle:
            pos: (0, 0)
            size: self.size
            radius: [25, ]
```

MDRelativeLayout

```
MDRelativeLayout:
    radius: [25, ]
    md_bg_color: app.theme_cls.primaryColor
```

API - kivymd.uix.relativelayout

class kivymd.uix.relativelayout.**MDRelativeLayout**(**args*, ***kwargs*)

Relative layout class.

For more information see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and `RelativeLayout` and *MDAdaptiveWidget* classes documentation.

2.3.8 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


class MobileView(MDScreen):
    pass


class TabletView(MDScreen):
    pass


class DesktopView(MDScreen):
    pass


class ResponsiveView(MDResponsiveLayout, MDScreen):
    def __init__(self, **kw):
        super().__init__(**kw)
```

(continues on next page)

(continued from previous page)

```
        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(**args*)

Called when the screen type changes.

on_size(**args*) → None

Called when the application screen size changes.

set_screen() → None

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

2.3.9 Hero

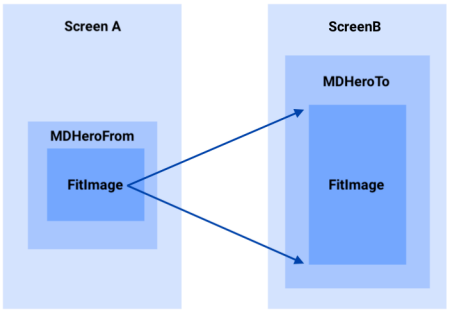
New in version 1.0.0.

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:

    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: "bg.jpg"
                size_hint: None, None
                size: hero_from.size

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["hero"]
                root.current = "screen B"

            MDButtonText:
                text: "Move Hero To Screen B"

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

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["hero"]
```

(continues on next page)

(continued from previous page)

```
                root.current = "screen A"

            MDButtonText:
                text: "Move Hero To Screen A"
'''


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


Example().run()
```

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

```
MDHeroFrom:
    id: hero_from
    tag: "hero"

    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:

```
MDButton:
    on_release:
        root.current_heroes = ["hero"]
        root.current = "screen 2"

    MDButtonText:
        text: "Move Hero To Screen B"
```

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):

```
MDButton:
    on_release:
        root.current_heroes = []
        root.current = "another screen"

    MDButtonText:
        text: "Go To 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

            FitImage:
                source: "bg.jpg"
                size_hint: None, None
                size: hero_from.size

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["hero"]
                root.current = "screen B"

            MDButtonText:
                text: "Move Hero To Screen B"

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

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "52dp"
            on_release:
                root.current_heroes = []
```

(continues on next page)

(continued from previous page)

```
                root.current = "screen C"

            MDButtonText:
                text: "Go To Screen C"

        MDButton:
            pos_hint: {"center_x": .5}
            y: "8dp"
            on_release:
                root.current_heroes = ["hero"]
                root.current = "screen A"

            MDButtonText:
                text: "Move Hero To Screen A"

    MDScreen:
        name: "screen C"
        md_bg_color: "olive"

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current = "screen B"

            MDButtonText:
                text: "Back To Screen B"
'''


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


Example().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
                md_bg_color: "blue"
                radius: [24, 12, 24, 12]

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["hero"]
                root.current = "screen B"

            MDButtonText:
                text: "Move Hero To Screen B"
```

(continues on next page)

(continued from previous page)

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

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["hero"]
                root.current = "screen A"

            MDButtonText:
                text: "Move Hero To Screen A"
'''


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


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

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

        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
    ):
        '''Fired when the hero back from screen **B** to screen **A**.'''
```

(continues on next page)

(continued from previous page)

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


Example().run()
```

Usage with ScrollView

```
from kivy.animation import Animation
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.metrics import dp
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: "24dp"

    MDSmartTile:
        id: tile
        size_hint: None, None
        size: root.size
        on_release: root.on_release()

        MDSmartTileImage:
            id: image
            source: "bg.jpg"
            radius: dp(24)

        MDSmartTileOverlayContainer:
            id: overlay
            md_bg_color: 0, 0, 0, .5
            adaptive_height: True
            padding: "8dp"
            spacing: "8dp"
            radius: [0, 0, dp(24), dp(24)]

            MDLabel:
                text: root.tag
                theme_text_color: "Custom"
                text_color: "white"
```

(continues on next page)

(continued from previous page)

```
                adaptive_height: True


MDScreenManager:
    md_bg_color: self.theme_cls.backgroundColor

    MDScreen:
        name: "screen A"

        ScrollView:

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

    MDScreen:
        name: "screen B"
        heroes_to: [hero_to]

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

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = [hero_to.tag]
                root.current = "screen A"

            MDButtonText:
                text: "Move Hero To Screen A"
'''


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

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

    def on_transform_in(self, instance_hero_widget, duration):
        for instance in [
            instance_hero_widget,
            instance_hero_widget._overlay_container,
```

(continues on next page)

(continued from previous page)

```
            instance_hero_widget._image,
        ]:
            Animation(radius=[0, 0, 0, 0], duration=duration).start(instance)

    def on_transform_out(self, instance_hero_widget, duration):
        for instance, radius in {
            instance_hero_widget: [dp(24), dp(24), dp(24), dp(24)],
            instance_hero_widget._overlay_container: [0, 0, dp(24), dp(24)],
            instance_hero_widget._image: [dp(24), dp(24), dp(24), dp(24)],
        }.items():
            Animation(
                radius=radius,
                duration=duration,
            ).start(instance)

    def on_release(self):
        def switch_screen(*args):
            self.manager.current_heroes = [self.tag]
            self.manager.ids.hero_to.tag = self.tag
            self.manager.current = "screen B"

        Clock.schedule_once(switch_screen, 0.2)


class Example(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)


Example().run()
```

Using multiple heroes at the same time

```
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_kivymd
            tag: "kivymd"
            size_hint: None, None
            size: "200dp", "200dp"
            pos_hint: {"top": .98}
            x: 24

            FitImage:
                source: "avatar.png"
                size_hint: None, None
                size: hero_kivymd.size
                radius: self.height / 2

        MDHeroFrom:
            id: hero_kivy
            tag: "kivy"
            size_hint: None, None
            size: "200dp", "200dp"
            pos_hint: {"top": .98}
            x: 324

            FitImage:
                source: "bg.jpg"
                size_hint: None, None
                size: hero_kivy.size
                radius: self.height / 2

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["kivymd", "kivy"]
                root.current = "screen B"

            MDButtonText:
                text: "Move Hero To Screen B"

    MDScreen:
        name: "screen B"
        heroes_to: hero_to_kivymd, hero_to_kivy
        md_bg_color: "cadetblue"

        MDHeroTo:
            id: hero_to_kivy
            tag: "kivy"
            size_hint: None, None
            pos_hint: {"center_x": .5, "center_y": .5}
```

(continues on next page)

(continued from previous page)

```
        MDHeroTo:
            id: hero_to_kivymd
            tag: "kivymd"
            size_hint: None, None
            pos_hint: {"right": 1, "top": 1}

        MDButton:
            pos_hint: {"center_x": .5}
            y: "36dp"
            on_release:
                root.current_heroes = ["kivy", "kivymd"]
                root.current = "screen A"

            MDButtonText:
                text: "Move Hero To Screen A"
'''


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


Example().run()
```

API - kivymd.uix.hero

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

The container from which the hero begins his flight.

For more information, see in the *MDBoxLayout* class documentation.

Events

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

on_transform_out
Fired when the hero back from screen **B** to screen **A**.

tag

Tag ID for heroes.

tag is an StringProperty and defaults to ‘’.

on_transform_in(**args*)

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

on_transform_out(**args*)

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

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

The container in which the hero comes.

For more information, see in the *MDBoxLayout* class documentation.

tag

Tag ID for heroes.

tag is an StringProperty and defaults to ‘’.

2.3.10 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDAnchorLayout

```
MDAnchorLayout:
    md_bg_color: app.theme_cls.primaryColor
```

API - kivymd.uix.anchorlayout

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

Anchor layout class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and AnchorLayout and *MDAdaptiveWidget* classes documentation.

2.3.11 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDRecycleGridLayout

```
MDRecycleGridLayout:
    adaptive_height: True
    md_bg_color: app.theme_cls.primaryColor
```

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 class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and RecycleGridLayout and *MDAdaptiveWidget* classes documentation.

2.3.12 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.

Transition

MDScreenManager class supports the following transitions:

- MDFadeSlideTransition
- MDSlideTransition
- MDSwapTransition

You need to use the *MDScreenManager* class when you want to use hero animations on your screens. If you don't need hero animation use the 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 *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and ScreenManager and *MDAdaptiveWidget* classes 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.

Deprecated since version 1.1.0: Use *current_heroes* attribute instead.

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

current_hero is an StringProperty and defaults to *None*.

current_heroes

A list of names (tags) of heroes that need to be animated when moving to the next screen.

New in version 1.1.0.

current_heroes is an ListProperty and defaults to *[]*.

check_transition(**args*) → None

Sets the default type transition.

get_hero_from_widget() → list

Get a list of *MDHeroFrom* objects according to the tag names specified in the *current_heroes* list.

on_current_hero(*instance*, *value: str*) → None

Fired when the value of the *current_hero* attribute changes.

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

Changed in version 2.1.0: Renamed argument *screen* to *widget*.

2.3.13 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.primaryColor
        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.primaryColor
```

API - kivymd.uix.widget

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

Widget class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and *MDAdaptiveWidget* and Widget and classes documentation.

New in version 1.0.0.

2.3.14 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.primaryColor
        Rectangle:
            pos: self.pos
            size: self.size
```

MDBoxLayout

```
MDBoxLayout:
    adaptive_height: True
    md_bg_color: app.theme_cls.primaryColor
```

Available options are:

- *adaptive_height*
- *adaptive_width*
- *adaptive_size*

adaptive_height

```
adaptive_height: True
```

Equivalent

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

adaptive_width

```
adaptive_width: True
```

Equivalent

```
size_hint_x: None
height: self.minimum_width
```

adaptive_size

```
adaptive_size: True
```

Equivalent

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

API - kivymd.uix.boxlayout

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

Box layout class.

For more information see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and BoxLayout and *MDAdaptiveWidget* classes documentation.

2.3.15 Screen

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

Screen

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

MDScreen

```
MDScreen:
    radius: [25, 0, 0, 0]
    md_bg_color: self.theme_cls.primaryColor
```

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 *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and `Screen` and *MDAdaptiveWidget* classes documentation.

hero_to

Must be a *MDHeroTo* class.

See the documentation of the MDHeroTo widget for more detailed information.

Deprecated since version 1.0.0: Use attr:*heroes_to* attribute instead.

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

heroes_to

Must be a list of *MDHeroTo* class.

New in version 1.0.0.

heroes_to is an `LiatProperty` and defaults to *[]*.

on_hero_to(*screen*, *widget:* kivymd.uix.hero.MDHeroTo) → None

Fired when the value of the *hero_to* attribute changes.

2.3.16 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 Example(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}"))


Example().run()
```

API - kivymd.uix.circularlayout

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

Float layout class. See module documentation for more information.

degree_spacing

The space between children in degree.

degree_spacing is an NumericProperty and defaults to *30*.

circular_radius

Radius of circle. Radius will be the greatest value in the layout if *circular_radius* if not specified.

circular_radius is an NumericProperty and defaults to *None*.

start_from

The positon of first child in degree.

start_from is an NumericProperty and defaults to *60*.

max_degree

Maximum range in degree allowed for each row of widgets before jumping to the next row.

max_degree is an NumericProperty and defaults to *360*.

circular_padding

Padding between outer widgets and the edge of the biggest circle.

circular_padding is an NumericProperty and defaults to *25dp*.

row_spacing

Space between each row of widget.

row_spacing is an NumericProperty and defaults to *50dp*.

clockwise

Direction of widgets in circular direction.

clockwise is an BooleanProperty and defaults to *True*.

get_angle(*pos: tuple*) → float

Returns the angle of given pos.

remove_widget(*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(**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.

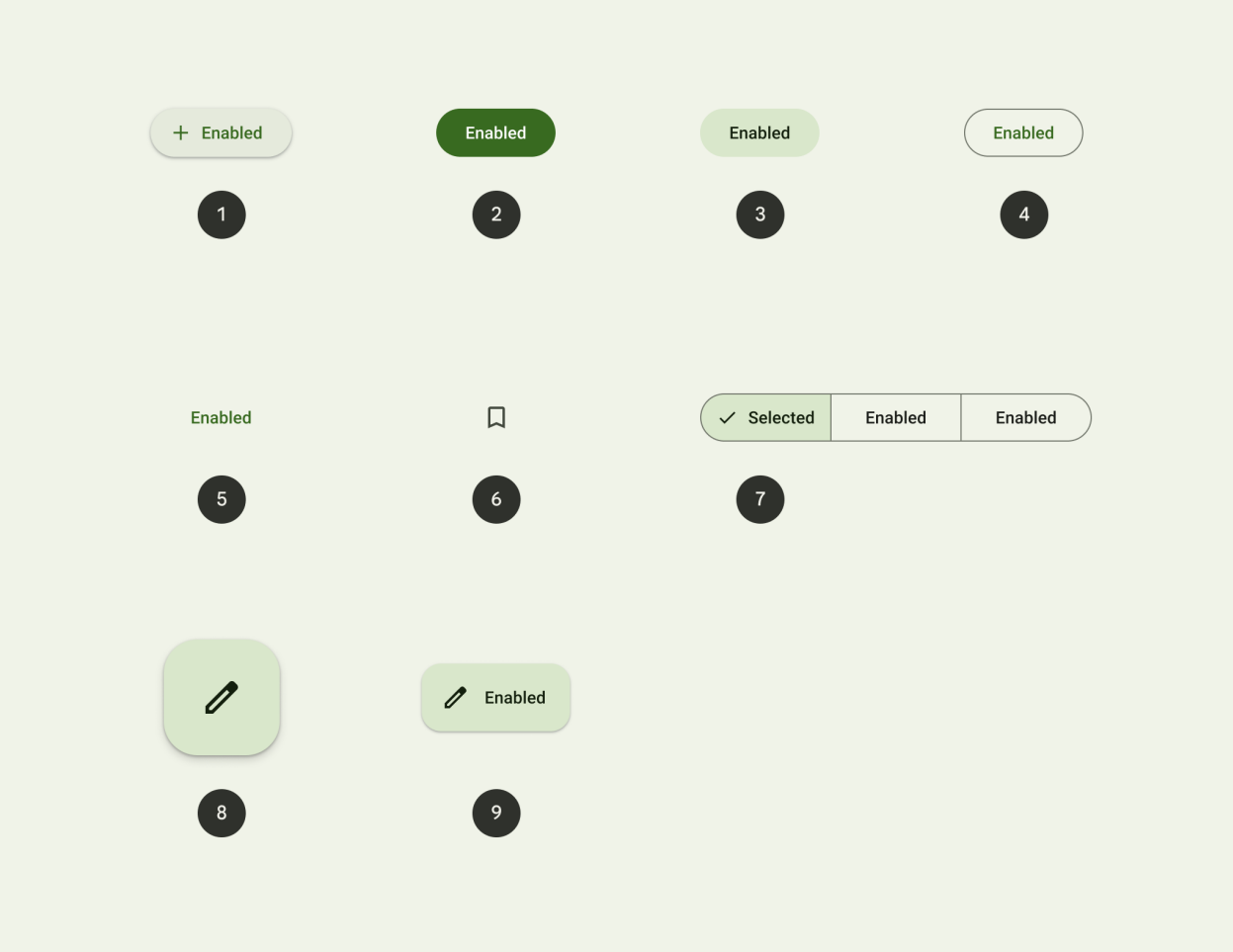
2.3.17 Button

See also:

Material Design spec, Buttons

Buttons allow users to take actions, and make choices, with a single tap. When choosing the right button for an action, consider the level of emphasis each button type provides.

KivyMD provides the following button classes for use:



1. Elevated button
2. Filled button
3. Filled tonal button
4. Outlined button
5. Text button
6. Icon button
7. Segmented button

8. Floating action button (FAB)
9. Extended FAB

Common buttons

Buttons help people take action, such as sending an email, sharing a document, or liking a comment.

1. Elevated button
2. Filled button
3. Filled tonal button
4. Outlined button
5. Text button

Elevated Filled Tonal Outlined Text

Elevated

Declarative KV style

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: app.theme_cls.surfaceColor

    MDButton:
        style: "elevated"
        pos_hint: {"center_x": .5, "center_y": .5}

        MDButtonIcon:
            icon: "plus"

        MDButtonText:
            text: "Elevated"
'''
```

(continues on next page)

(continued from previous page)

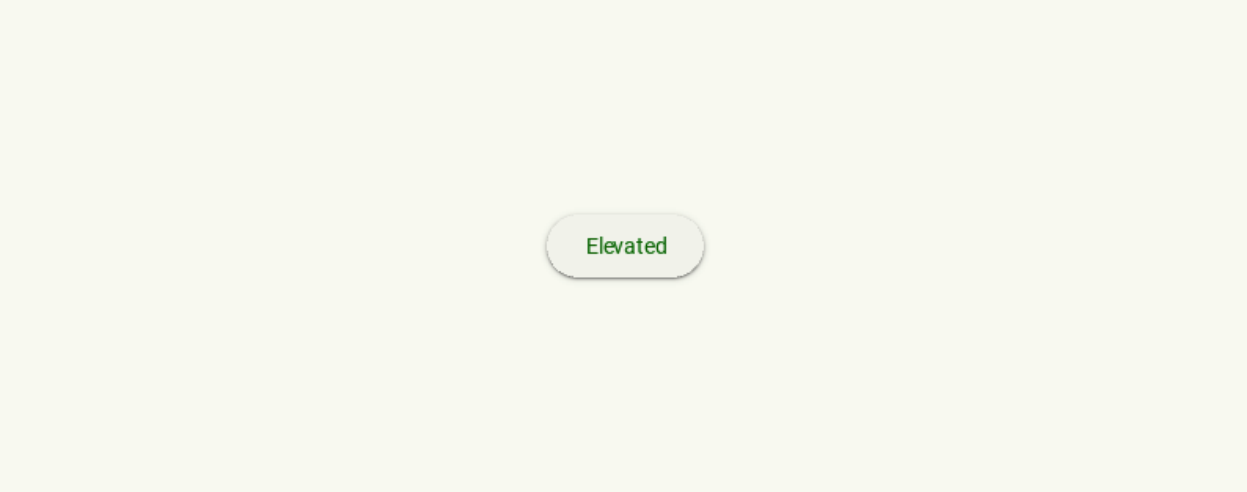
```
class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Green"
        return Builder.load_string(KV)


Example().run()
```

Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonIcon, MDButtonText
from kivymd.uix.screen import MDScreen


class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Green"
        return (
            MDScreen(
                MDButton(
                    MDButtonIcon(
                        icon="plus",
                    ),
                    MDButtonText(
                        text="Elevated",
                    ),
                    style="elevated",
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                ),
                md_bg_color=self.theme_cls.surfaceColor,
            )
        )


Example().run()
```

Common buttons can contain an icon or be without an icon:

```
MDButton:
    style: "elevated"
    text: "Elevated"
```



Filled

```
MDButton:
    style: "filled"

    MDButtonText:
        text: "Filled"
```

Tonal

```
MDButton:
    style: "tonal"

    MDButtonText:
        text: "Tonal"
```

Outlined

```
MDButton:
    style: "outlined"

    MDButtonText:
        text: "Outlined"
```

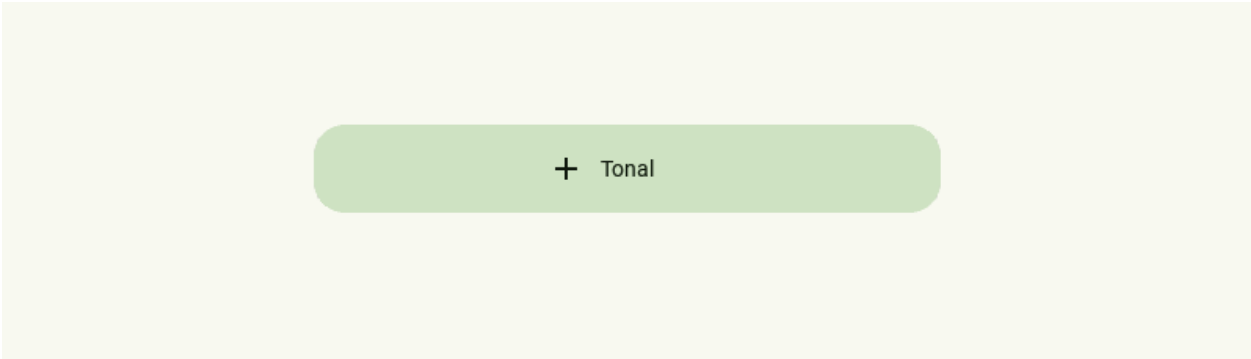
Text

```
MDButton:
    style: "text"

    MDButtonText:
        text: "Text"
```

Customization of buttons

Text positioning and button size

```
MDButton:
    style: "tonal"
    theme_width: "Custom"
    height: "56dp"
    size_hint_x: .5

    MDButtonIcon:
        x: text.x - (self.width + dp(10))
        icon: "plus"

    MDButtonText:
        id: text
        text: "Tonal"
        pos_hint: {"center_x": .5, "center_y": .5}
```



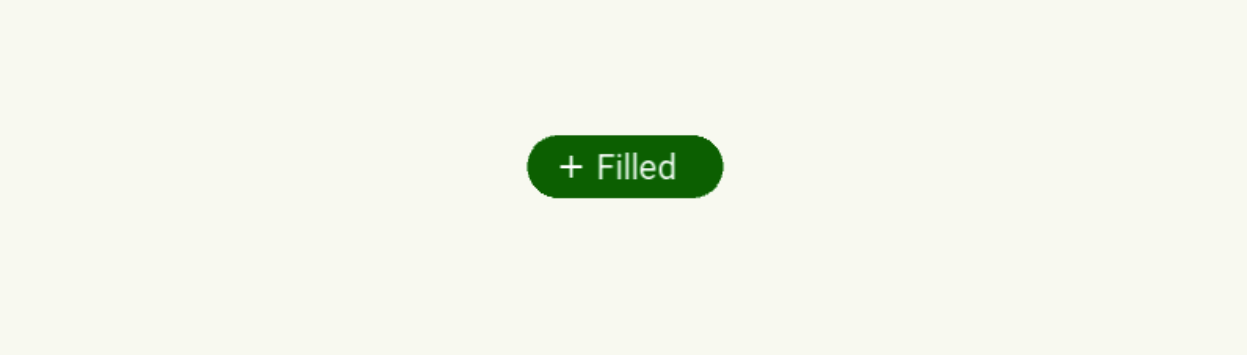
Font of the button text

```
MDButton:
    style: "filled"

    MDButtonIcon:
        icon: "plus"

    MDButtonText:
```

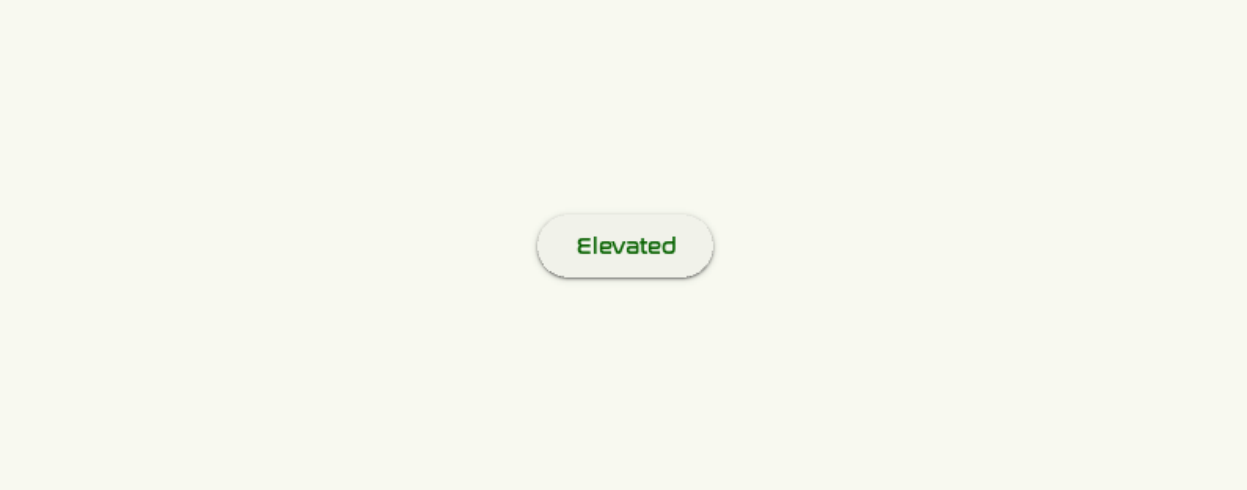
(continues on next page)

(continued from previous page)

```
        text: "Filled"
        font_style: "Title"
```




```
MDButton:
    style: "elevated"

    MDButtonText:
        text: "Elevated"
        theme_font_name: "Custom"
        font_name: "path/to/font.ttf"
```



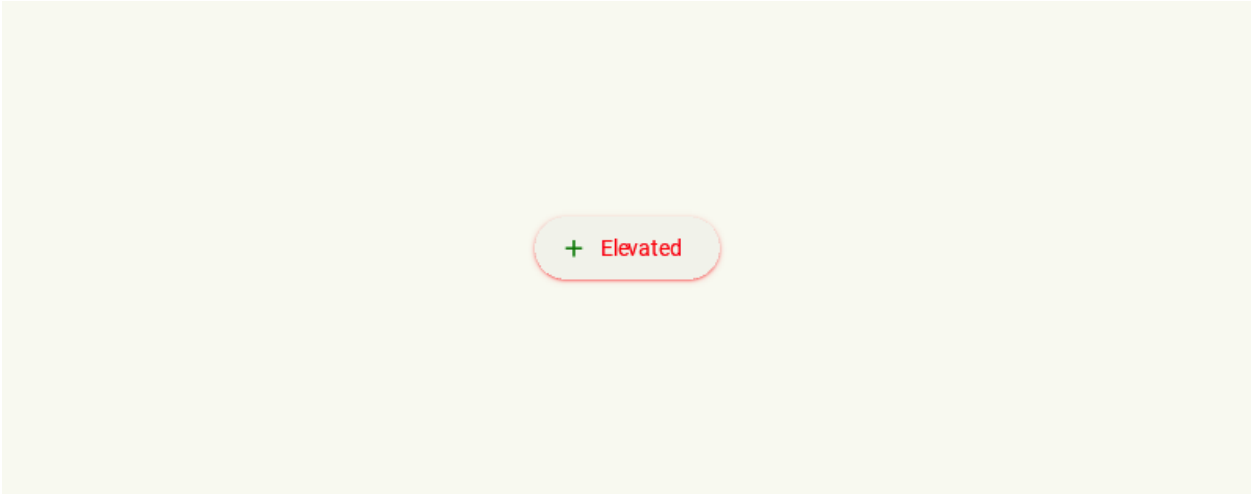
Custom button color

```
MDButton:
    style: "elevated"
    theme_shadow_color: "Custom"
    shadow_color: "red"

    MDButtonIcon:
        icon: "plus"
        theme_icon_color: "Custom"
        icon_color: "green"
```

(continues on next page)

(continued from previous page)

```
    MDButtonText:
        text: "Elevated"
        theme_text_color: "Custom"
        text_color: "red"
```

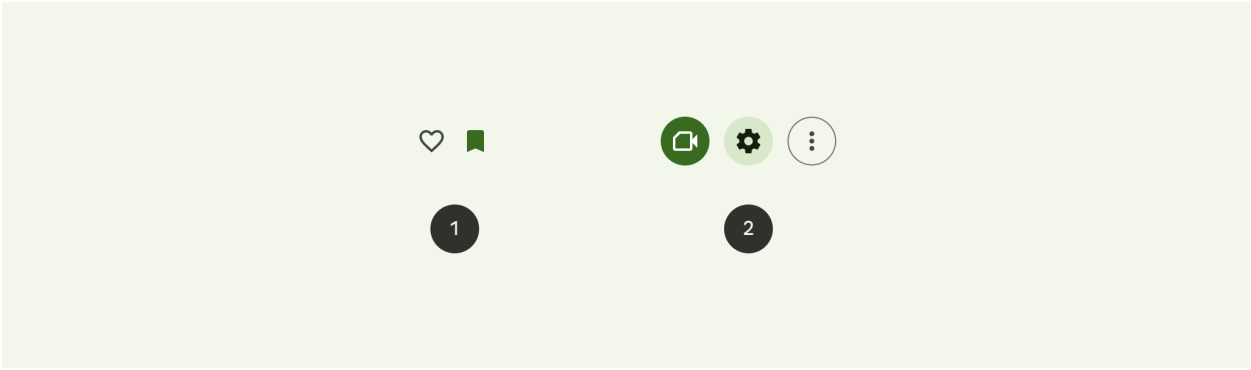


Icon buttons

Use icon buttons when a compact button is required, such as in a toolbar or image list. There are two types of icon buttons: standard and contained.

See also:

Material Design spec, Icon buttons



1. Standard icon button
2. Contained icon button (including filled, filled tonal, and outlined styles)

StandardIcon FilledIcon TonalIcon OutlinedIcon

StandardIcon

```
MDIconButton:
    icon: "heart-outline"
    style: "standard"
```

FilledIcon

```
MDIconButton:
    icon: "heart-outline"
    style: "filled"
```

TonalIcon

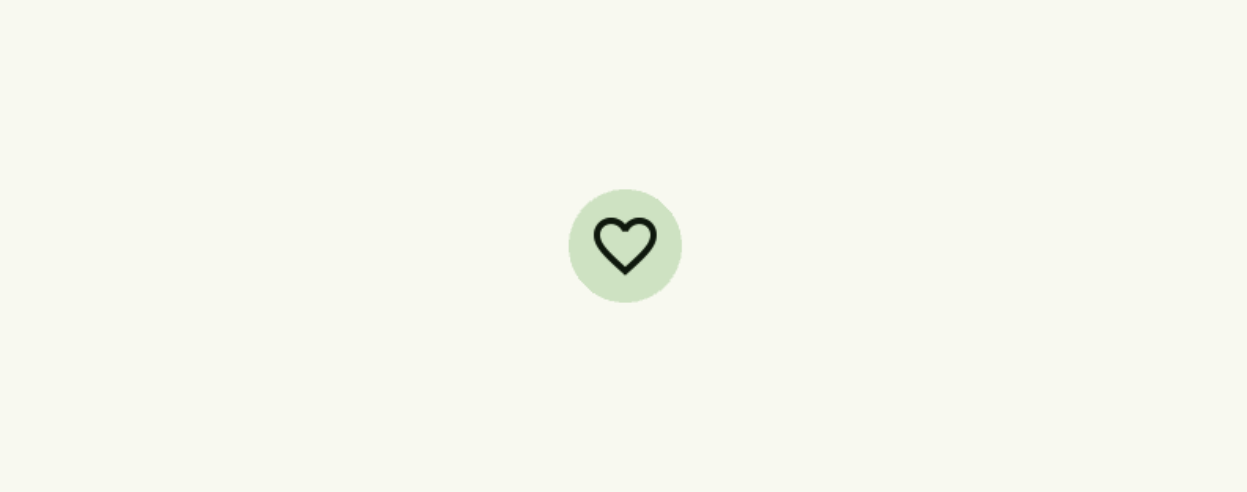
```
MDIconButton:
    icon: "heart-outline"
    style: "tonal"
```

OutlinedIcon

```
MDIconButton:
    icon: "heart-outline"
    style: "outlined"
```

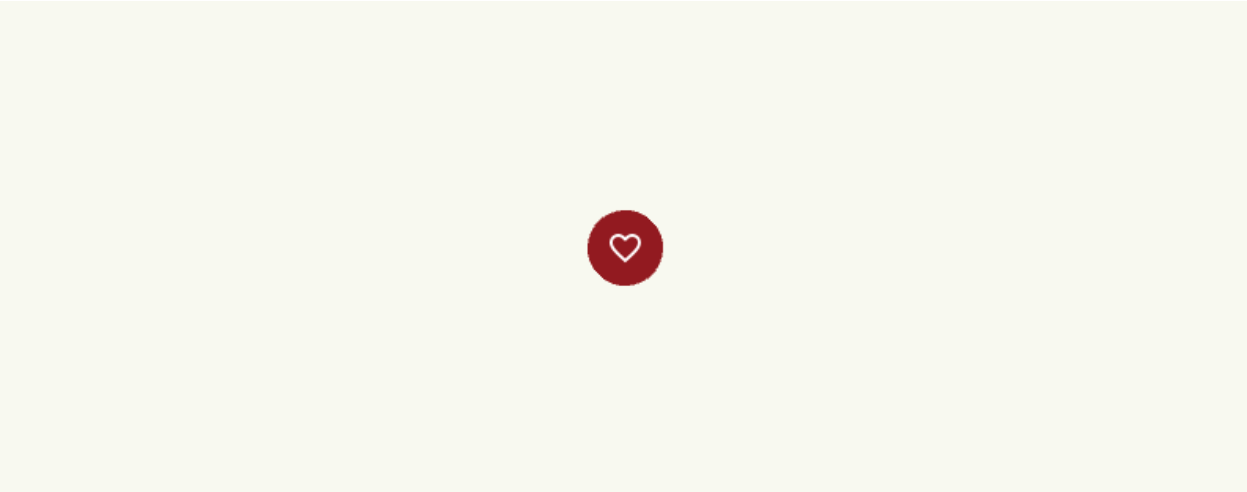
Custom icon size

```
MDIconButton:
    icon: "heart-outline"
    style: "tonal"
    theme_font_size: "Custom"
    font_size: "48sp"
    radius: [self.height / 2, ]
    size_hint: None, None
    size: "84dp", "84dp"
```



Custom button color

```
MDIconButton:
    icon: "heart-outline"
    style: "tonal"
    theme_bg_color: "Custom"
    md_bg_color: "brown"
    theme_icon_color: "Custom"
    icon_color: "white"
```

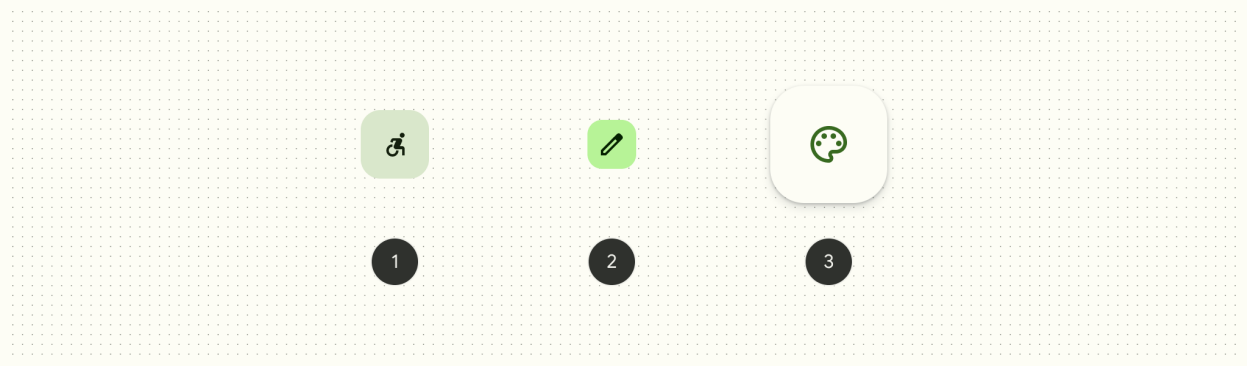


FAB buttons

The FAB represents the most important action on a screen. It puts key actions within reach.

See also:

Material Design spec, FAB buttons



1. Standard FAB
2. Small FAB
3. Large FAB

There are three sizes of floating action buttons: FAB, small FAB, and large FAB:

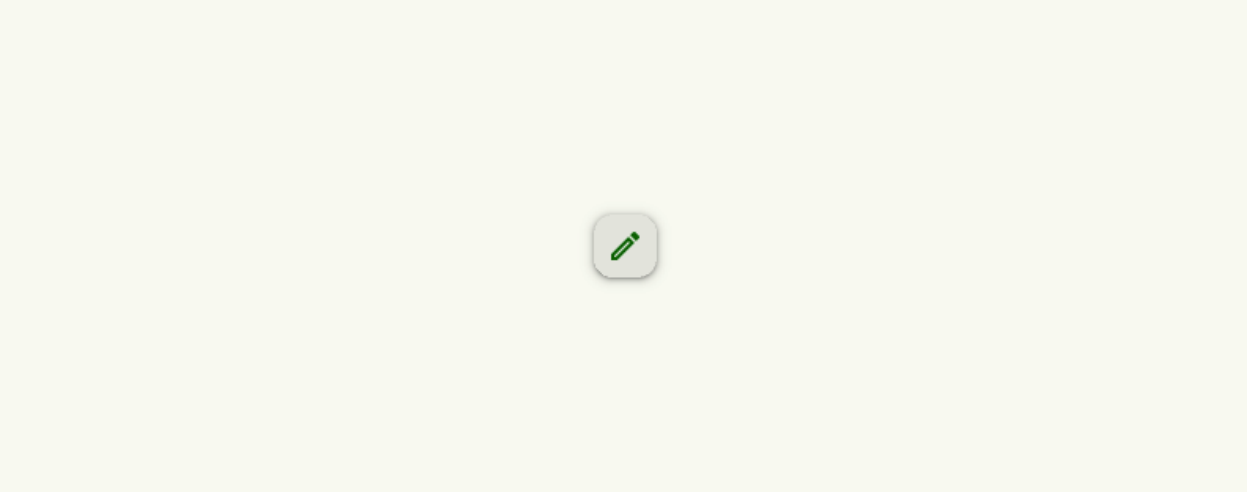
Standard Small Large

Standard

```
MDFabButton:
    icon: "pencil-outline"
    style: "standard"
```

Small

```
MDFabButton:
    icon: "pencil-outline"
    style: "small"
```



Large

```
MDFabButton:
    icon: "pencil-outline"
    style: "large"
```

Additional color mappings

FABs can use other combinations of container and icon colors. The color mappings below provide the same legibility and functionality as the default, so the color mapping you use depends on style alone.

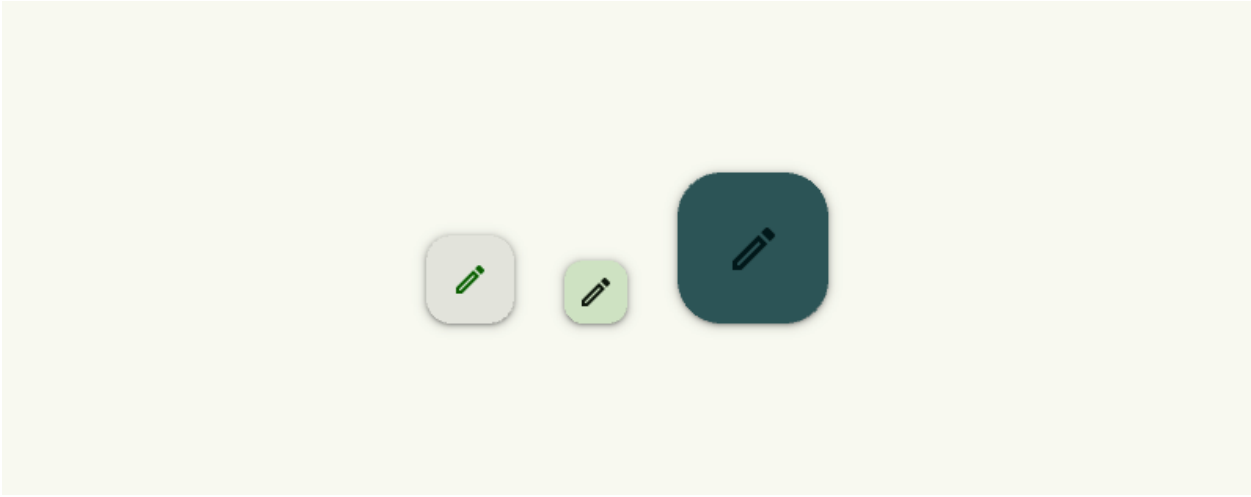
1. Surface
2. Secondary
3. Tertiary

```
from kivy.lang import Builder

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

```

(continues on next page)

(continued from previous page)

```
KV = '''
MDScreen:
    md_bg_color: app.theme_cls.surfaceColor

    MDBoxLayout:
        id: box
        adaptive_size: True
        spacing: "32dp"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


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

    def on_start(self):
        styles = {
            "standard": "surface",
            "small": "secondary",
            "large": "tertiary",
        }
        for style in styles.keys():
            self.root.ids.box.add_widget(
                MDFabButton(
                    style=style, icon="pencil-outline", color_map=styles[style]
                )
            )


Example().run()
```

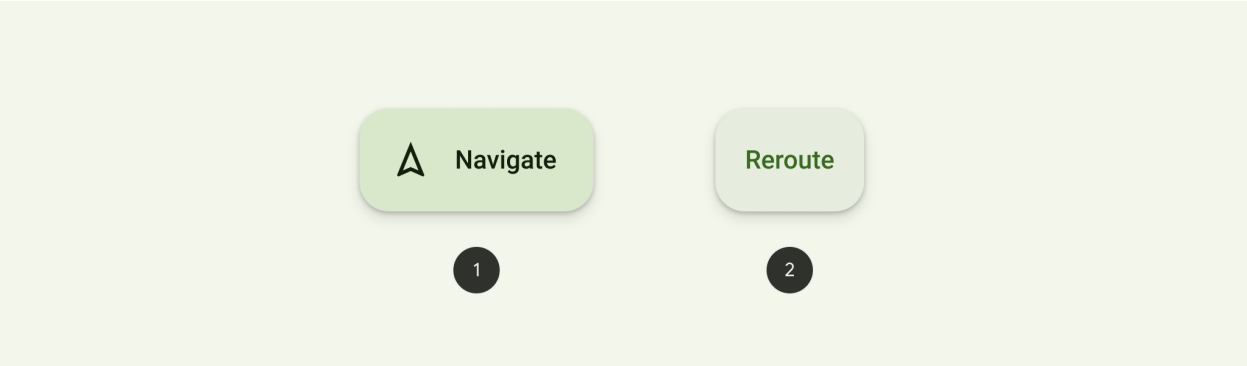


Extended FAB

Extended floating action buttons (extended FABs) help people take primary actions. They're wider than FABs to accommodate a text label and larger target area.

See also:

Material Design spec, FAB extended buttons



1. Extended FAB with both icon and label text
2. Extended FAB without icon

With icon

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: app.theme_cls.surfaceColor
    on_touch_down:
        if not btn.collide_point(*args[1].pos): \
        btn.fab_state = "expand" \
        if btn.fab_state == "collapse" else "collapse"

        MDExtendedFabButton:
            id: btn
            pos_hint: {"center_x": .5, "center_y": .5}

            MDExtendedFabButtonIcon:
                icon: "pencil-outline"

            MDExtendedFabButtonText:
                text: "Compose"
'''


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

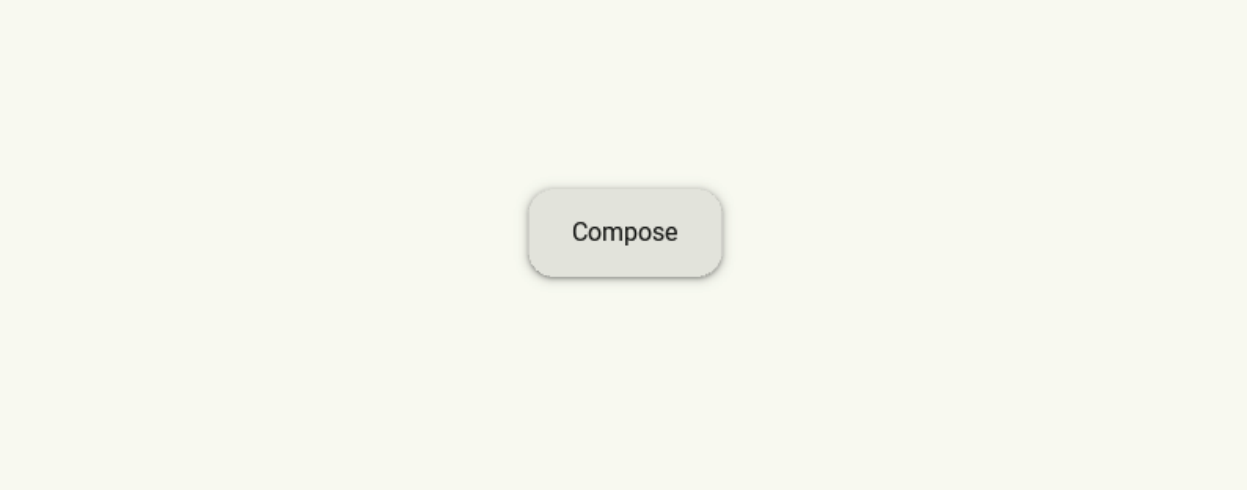
(continues on next page)

(continued from previous page)

```
        self.theme_cls.primary_palette = "Green"
        return Builder.load_string(KV)


Example().run()
```

Without icon

```
MDExtendedFabButton:
    fab_state: "expand"

    MDExtendedFabButtonText:
        text: "Compose"
```



API break

1.2.0 version

```
MDFloatingActionButton:
    icon: "plus"
```

```
MDRoundFlatButton:
    text: "Outlined"
```

```
MDRoundFlatIconButton:
    text: "Outlined with icon"
    icon: "plus"
```

```
MDFillRoundFlatButton
    text: "Filled"
```

```
MDFillRoundFlatIconButton
    text: "Filled with icon"
    icon: "plus"
```

2.0.0 version

Note: *MDFloatingActionButtonSpeedDial* type buttons were removed in version *2.0.0*.

```
MDFabButton:
    icon: "plus"
```

```
MDButton:
    style: "outlined"

    MDButtonText:
        text: "Outlined"
```

```
MDButton:
    style: "outlined"

    MDButtonIcon:
        icon: "plus"

    MDButtonText:
        text: "Outlined with icon"
```

```
MDButton:
    style: "filled"

    MDButtonText:
        text: "Filled"
```

```
MDButton:
    style: "filled"

    MDButtonIcon:
        icon: "plus"

    MDButtonText:
        text: "Filled"
```

API - kivymd.uix.button.button

class kivymd.uix.button.button.**BaseFabButton**

Implements the basic properties for the *MDExtendedFabButton* and *MDFabButton* classes.

New in version 2.0.0.

elevation_levels

Elevation is measured as the distance between components along the z-axis in density-independent pixels (dps).

New in version 1.2.0.

elevation_levels is an DictProperty and defaults to *{0: dp(0), 1: dp(4), 2: dp(8), 3: dp(12), 4: dp(16), 5: dp(18)}*.

color_map

Additional color mappings.

Available options are: 'surface', 'secondary', 'tertiary'.

color_map is an OptionProperty and defaults to *'secondary'*.

icon_color_disabled

The icon color in (r, g, b, a) or string format of the list item when the widget item is disabled.

icon_color_disabled is a ColorProperty and defaults to *None*.

style

Button type.

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

style is an OptionProperty and defaults to *'standard'*.

fab_state

The state of the button.

Available options are: 'collapse' or 'expand'.

fab_state is an OptionProperty and defaults to "collapse".

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the list item when the list button is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

radius

Canvas radius.

radius is an VariableListProperty and defaults to *[dp(16), dp(16), dp(16), dp(16)]*.

class kivymd.uix.button.button.**BaseButton**(**args*, ***kwargs*)

Base button class.

For more information, see in the *DeclarativeBehavior* and *BackgroundColorBehavior* and *RectangularRippleBehavior* and ButtonBehavior and *ThemableBehavior* and *StateLayerBehavior* classes documentation.

elevation_levels

Elevation is measured as the distance between components along the z-axis in density-independent pixels (dps).

New in version 1.2.0.

elevation_levels is an DictProperty and defaults to *{0: dp(0), 1: dp(4), 2: dp(8), 3: dp(12), 4: dp(16), 5: dp(18)}*.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the button when the button is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

shadow_radius

Button shadow radius.

shadow_radius is an VariableListProperty and defaults to *[0, 0, 0, 0]*.

md_bg_color

Button background color in (r, g, b, a) or string format.

md_bg_color is a ColorProperty and defaults to *None*.

line_color

Outlined color.

line_color is a ColorProperty and defaults to *None*.

line_width

Line width for button border.

line_width is a NumericProperty and defaults to *1*.

on_press(**args*) → None

Fired when the button is pressed.

on_release(**args*) → None

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

class kivymd.uix.button.button.**MDButton**(**args*, ***kwargs*)

Base class for all buttons.

New in version 2.2.0.

For more information, see in the *CommonElevationBehavior* and *BaseButton* and RelativeLayout classes documentation.

style

Button type.

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

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

radius

Button radius.

radius is an VariableListProperty and defaults to *[dp(20), dp(20), dp(20), dp(20)]*.

adjust_pos(**args*) → None

Adjusts the pos of the button according to the content.

adjust_width(**args*) → None

Adjusts the width of the button according to the content.

add_widget(*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)
```

set_properties_widget() → None

Fired *on_release/on_press/on_enter/on_leave* events.

class kivymd.uix.button.button.**MDButtonText**(**args*, ***kwargs*)

The class implements the text for the *MDButton* class.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.button.button.**MDButtonIcon**(**args*, ***kwargs*)

The class implements an icon for the *MDButton* class.

For more information, see in the *MDIcon* class documentation.

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

Base class for icon buttons.

For more information, see in the *RectangularRippleBehavior* and ButtonBehavior and MDIcon classes documentation.

style

Button type.

New in version 2.0.0.

Available options are: 'standard', 'filled', 'tonal', 'outlined'.

style is an OptionProperty and defaults to *'standard'*.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the list item when the list button is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

on_line_color(*instance*, *value*) → None

Fired when the values of `line_color` change.

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

Base class for FAB buttons.

For more information, see in the *BaseFabButton* and *CommonElevationBehavior* and *RectangularRippleBehavior* and ButtonBehavior and *MDIcon* classes documentation.

on_press(**args*) → None

Fired when the button is pressed.

on_release(**args*) → None

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

set_properties_widget() → None

Fired *on_release/on_press/on_enter/on_leave* events.

class kivymd.uix.button.button.**MDExtendedFabButtonIcon**(**args*, ***kwargs*)

Implements an icon for the *MDExtendedFabButton* class.

New in version 2.0.0.

class kivymd.uix.button.button.**MDExtendedFabButtonText**(**args*, ***kwargs*)

Implements the text for the class *MDExtendedFabButton* class.

New in version 2.0.0.

class kivymd.uix.button.button.**MDExtendedFabButton**(**args*, ***kwargs*)

Base class for Extended FAB buttons.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *MotionExtendedFabButtonBehavior* and *CommonElevationBehavior* and *StateLayerBehavior* and *BaseFabButton* and ButtonBehavior and RelativeLayout classes documentation.

Events

on_collapse
Fired when the button is collapsed.

on_expand
Fired when the button is expanded.

elevation_levels

Elevation is measured as the distance between components along the z-axis in density-independent pixels (dps).

New in version 1.2.0.

elevation_levels is an DictProperty and defaults to *{0: dp(0), 1: dp(4), 2: dp(8), 3: dp(12), 4: dp(16), 5: dp(18)}*.

add_widget(*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_collapse(**args*)

Fired when the button is collapsed.

on_expand(**args*)

Fired when the button is expanded.

on_fab_state(*instance*, *state: str*) → None

Fired when the `fab_state` value changes.

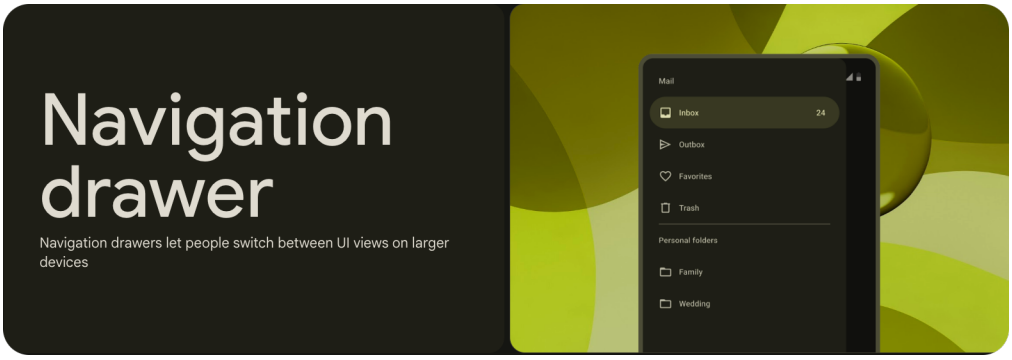
on__x(*instance*, *value*) → None

2.3.18 NavigationDrawer

See also:

Material Design, Navigation drawer

Navigation drawers provide access to destinations in your app.



- Use navigation drawers in expanded layouts and modal navigation drawers in compact and medium layouts
- Can be open or closed by default
- Two types: standard and modal

When using the class *MDNavigationDrawer* skeleton of your *KV* markup should look like this:

Usage

```
Root:

    MDNavigationLayout:

        MDScreenManager:

            Screen_1:

            Screen_2:

        MDNavigationDrawer:

            # This custom rule should implement what will be displayed in
            # your MDNavigationDrawer.
            ContentNavigationDrawer:
```

A simple example

Declarative KV styles

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

```

(continues on next page)

(continued from previous page)

```
    MDNavigationLayout:

        MDScreenManager:

            MDScreen:

                MDButton:
                    pos_hint: {"center_x": .5, "center_y": .5}
                    on_release: nav_drawer.set_state("toggle")

                    MDButtonText:
                        text: "Open Drawer"

        MDNavigationDrawer:
            id: nav_drawer
            radius: 0, dp(16), dp(16), 0

            MDNavigationDrawerMenu:

                MDNavigationDrawerLabel:
                    text: "Mail"

                MDNavigationDrawerItem:

                    MDNavigationDrawerItemLeadingIcon:
                        icon: "account"

                    MDNavigationDrawerItemText:
                        text: "Inbox"

                    MDNavigationDrawerItemTrailingText:
                        text: "24"

                MDNavigationDrawerDivider:
'''


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


Example().run()
```

Declarative python styles

```
from kivy.metrics import dp

from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.screenmanager import MDScreenManager
from kivymd.uix.navigationdrawer import (
    MDNavigationLayout,
```

(continues on next page)

(continued from previous page)

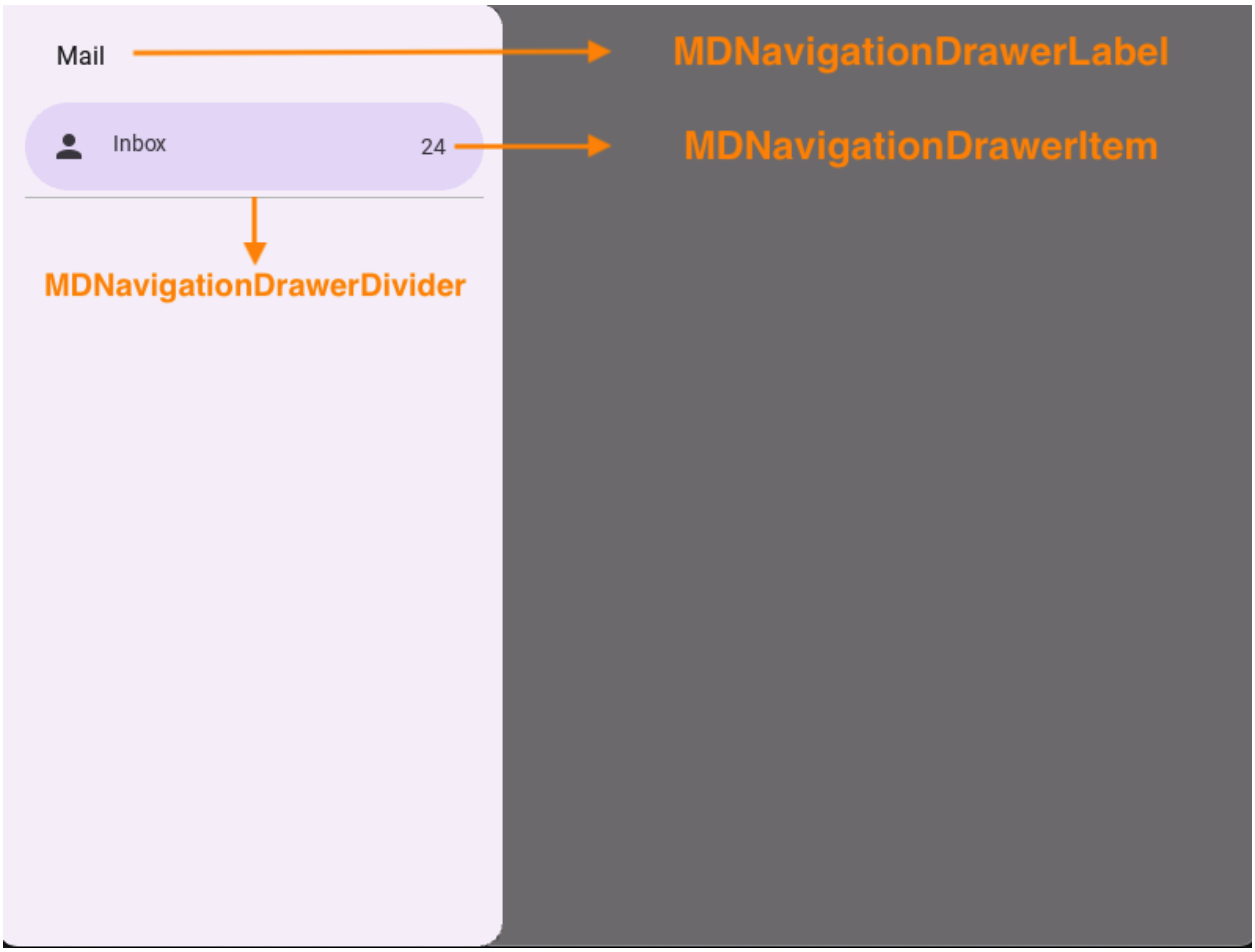
```
    MDNavigationDrawer,
    MDNavigationDrawerMenu,
    MDNavigationDrawerLabel,
    MDNavigationDrawerItem,
    MDNavigationDrawerItemLeadingIcon,
    MDNavigationDrawerItemText,
    MDNavigationDrawerItemTrailingText,
    MDNavigationDrawerDivider,
)
from kivymd.uix.screen import MDScreen
from kivymd.app import MDApp


class Example(MDApp):
    def build(self):
        return MDScreen(
            MDNavigationLayout(
                MDScreenManager(
                    MDScreen(
                        MDButton(
                            MDButtonText(
                                text="Open Drawer",
                            ),
                            on_release=lambda x: self.root.get_ids().nav_drawer.set_
↪state(
                                "toggle"
                            ),
                            pos_hint={"center_x": 0.5, "center_y": 0.5},
                        ),
                    ),
                ),
                MDNavigationDrawer(
                    MDNavigationDrawerMenu(
                        MDNavigationDrawerLabel(
                            text="Mail",
                        ),
                        MDNavigationDrawerItem(
                            MDNavigationDrawerItemLeadingIcon(
                                icon="account",
                            ),
                            MDNavigationDrawerItemText(
                                text="Inbox",
                            ),
                            MDNavigationDrawerItemTrailingText(
                                text="24",
                            ),
                        ),
                        MDNavigationDrawerDivider(
                        ),
                    ),
                    id="nav_drawer",
                    radius=(0, dp(16), dp(16), 0),
```

(continues on next page)

(continued from previous page)

```
                ),
            ),
            md_bg_color=self.theme_cls.backgroundColor,
        )


Example().run()
```

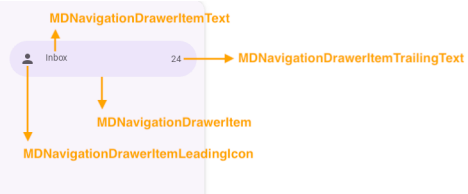
Anatomy



Note: *MDNavigationDrawer* is an empty `MDCard` panel.

Item anatomy

```
MDNavigationDrawerItem:

    MDNavigationDrawerItemLeadingIcon:
        icon: "account"

    MDNavigationDrawerItemText:
        text: "Inbox"

    MDNavigationDrawerItemTrailingText:
        text: "24"
```



Type drawer

Standard

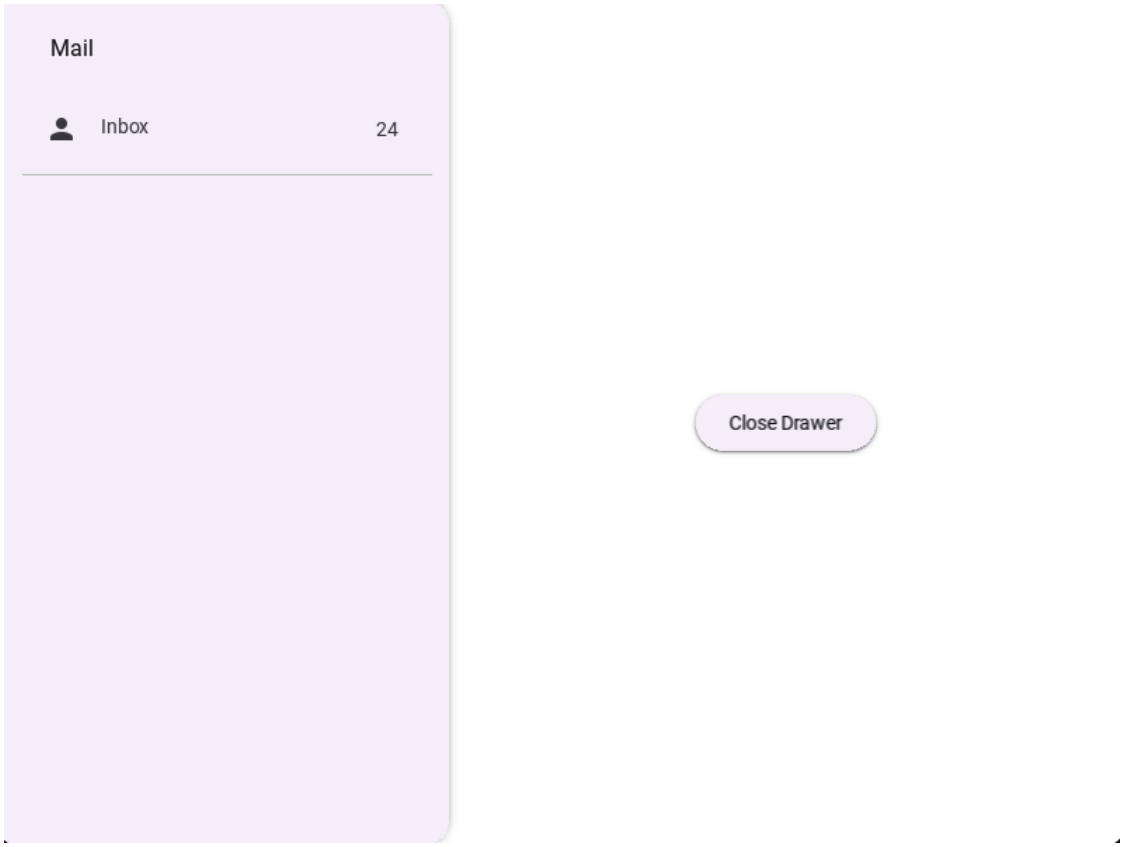
```
MDNavigationDrawer:
    drawer_type: "standard"
```

Modal

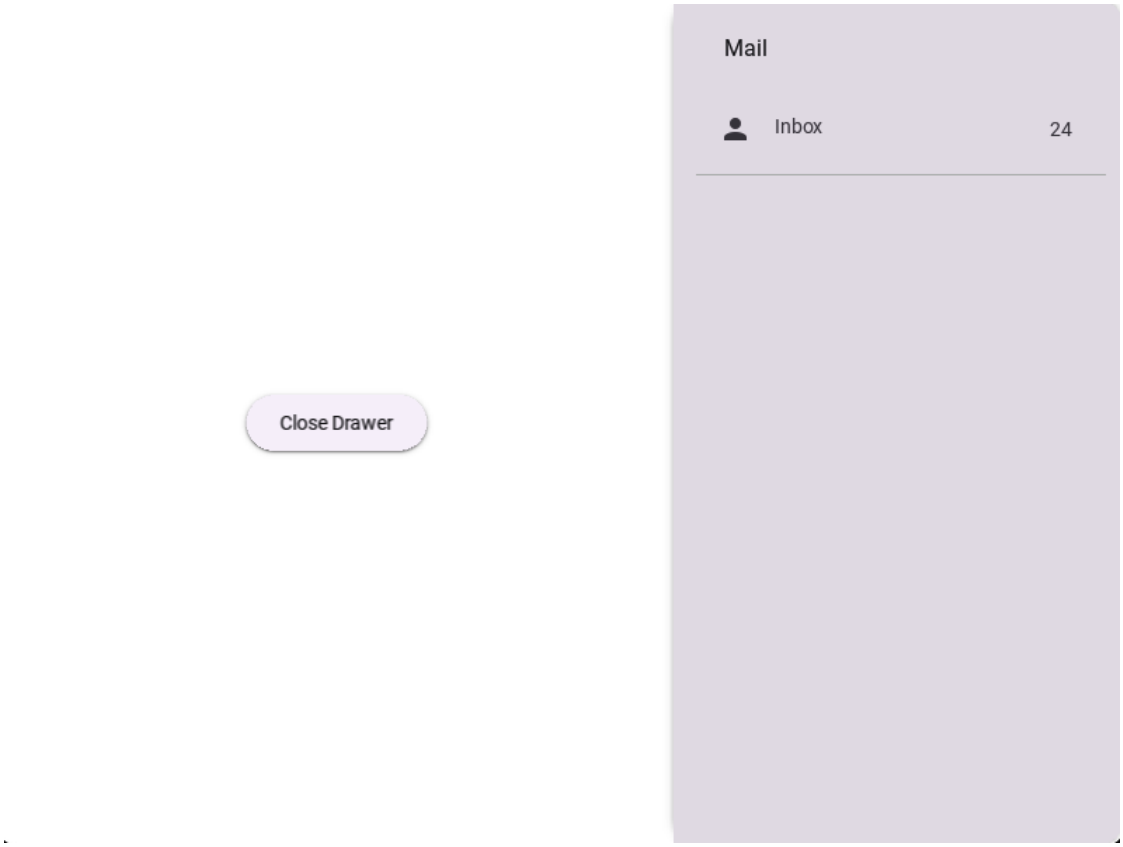
```
MDNavigationDrawer:
    drawer_type: "modal"
```

Anchoring screen edge for drawer

```
MDNavigationDrawer:
    anchor: "left"
```




```
MDNavigationDrawer:
    anchor: "right"
```



API break

1.2.0 version

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<DrawerClickableItem@MDNavigationDrawerItem>
    focus_color: "#e7e4c0"
    text_color: "#4a4939"
    icon_color: "#4a4939"
    ripple_color: "#c5bdd2"
    selected_color: "#0c6c4d"


<DrawerLabelItem@MDNavigationDrawerItem>
    text_color: "#4a4939"
    icon_color: "#4a4939"
    focus_behavior: False
    selected_color: "#4a4939"
    _no_ripple_effect: True
```

(continues on next page)

(continued from previous page)

```
MDScreen:

    MDNavigationLayout:

        MDScreenManager:

            MDScreen:

                MDRaisedButton:
                    text: "Open Drawer"
                    pos_hint: {"center_x": .5, "center_y": .5}
                    on_release: nav_drawer.set_state("toggle")

        MDNavigationDrawer:
            id: nav_drawer
            radius: (0, dp(16), dp(16), 0)

            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:
                    icon: "send"
                    text: "Outbox"

                MDNavigationDrawerDivider:

                MDNavigationDrawerLabel:
                    text: "Labels"

                DrawerLabelItem:
                    icon: "information-outline"
                    text: "Label"

                DrawerLabelItem:
                    icon: "information-outline"
                    text: "Label"
```

(continues on next page)

(continued from previous page)

```
'''


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


Example().run()
```

2.2.0 version

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

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.navigationdrawer import (
    MDNavigationDrawerItem, MDNavigationDrawerItemTrailingText
)

KV = '''
<DrawerItem>
    active_indicator_color: "#e7e4c0"

    MDNavigationDrawerItemLeadingIcon:
        icon: root.icon
        theme_icon_color: "Custom"
        icon_color: "#4a4939"

    MDNavigationDrawerItemText:
        text: root.text
        theme_text_color: "Custom"
        text_color: "#4a4939"


<DrawerLabel>
    adaptive_height: True
    padding: "18dp", 0, 0, "12dp"

    MDNavigationDrawerItemLeadingIcon:
        icon: root.icon
        theme_icon_color: "Custom"
        icon_color: "#4a4939"
        pos_hint: {"center_y": .5}

    MDNavigationDrawerLabel:
        text: root.text
        theme_text_color: "Custom"
        text_color: "#4a4939"
```

(continues on next page)

(continued from previous page)

```
        pos_hint: {"center_y": .5}
        padding: "6dp", 0, "16dp", 0
        theme_line_height: "Custom"
        line_height: 0


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDNavigationLayout:

        MDScreenManager:

            MDScreen:

                MDButton:
                    pos_hint: {"center_x": .5, "center_y": .5}
                    on_release: nav_drawer.set_state("toggle")

                    MDButtonText:
                        text: "Open Drawer"

        MDNavigationDrawer:
            id: nav_drawer
            radius: 0, dp(16), dp(16), 0

            MDNavigationDrawerMenu:

                MDNavigationDrawerHeader:
                    orientation: "vertical"
                    padding: 0, 0, 0, "12dp"
                    adaptive_height: True

                    MDLabel:
                        text: "Header title"
                        theme_text_color: "Custom"
                        theme_line_height: "Custom"
                        line_height: 0
                        text_color: "#4a4939"
                        adaptive_height: True
                        padding_x: "16dp"
                        font_style: "Display"
                        role: "small"

                    MDLabel:
                        text: "Header text"
                        padding_x: "18dp"
                        adaptive_height: True
                        font_style: "Title"
                        role: "large"

                MDNavigationDrawerDivider:
```

(continues on next page)

(continued from previous page)

```
                DrawerItem:
                    icon: "gmail"
                    text: "Inbox"
                    trailing_text: "+99"
                    trailing_text_color: "#4a4939"

                DrawerItem:
                    icon: "send"
                    text: "Outbox"

                MDNavigationDrawerDivider:

                MDNavigationDrawerLabel:
                    text: "Labels"
                    padding_y: "12dp"

                DrawerLabel:
                    icon: "information-outline"
                    text: "Label"

                DrawerLabel:
                    icon: "information-outline"
                    text: "Label"
'''


class DrawerLabel(MDBoxLayout):
    icon = StringProperty()
    text = StringProperty()


class DrawerItem(MDNavigationDrawerItem):
    icon = StringProperty()
    text = StringProperty()
    trailing_text = StringProperty()
    trailing_text_color = ColorProperty()

    _trailing_text_obj = None

    def on_trailing_text(self, instance, value):
        self._trailing_text_obj = MDNavigationDrawerItemTrailingText(
            text=value,
            theme_text_color="Custom",
            text_color=self.trailing_text_color,
        )
        self.add_widget(self._trailing_text_obj)

    def on_trailing_text_color(self, instance, value):
        self._trailing_text_obj.text_color = value
```

(continues on next page)

(continued from previous page)

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


Example().run()
```

API - kivymd.uix.navigationdrawer.navigationdrawer

class kivymd.uix.navigationdrawer.navigationdrawer.**BaseNavigationDrawerItem**

Implement the base class for the menu list item.

New in version 2.0.0.

selected

Is the item selected.

selected is a BooleanProperty and defaults to *False*.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationLayout**(**args*, ***kwargs*)

For more information, see in the *DeclarativeBehavior* and FloatLayout classes documentation.

update_pos(*instance_navigation_drawer*, *pos_x: float*) → None

add_scrim(*instance_manager: kivy.uix.screenmanager.ScreenManager*) → None

update_scrim_rectangle(*instance_manager: kivy.uix.screenmanager.ScreenManager*, *size: list*) → None

add_widget(*widget*, *index=0*, *canvas=None*)

Only two layouts are allowed: ScreenManager and *MDNavigationDrawer*.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerLabel**(**args*, ***kwargs*)

Implements a label class.

For more information, see in the *MDLabel* class documentation.

New in version 1.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerDivider**(***kwargs*)

Implements a divider class.

For more information, see in the BoxLayout class documentation.

New in version 1.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerHeader**(**args*, ***kwargs*)

Implements a header class.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

New in version 1.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerItem**(**args*, ***kwargs*)

Implements an item for the *MDNavigationDrawer* menu list.

For more information, see in the *MDListItem* and *FocusBehavior* and *BaseNavigationDrawerItem* classes documentation.

New in version 1.0.0.

active_indicator_color

The active indicator color in (r, g, b, a) or string format.

New in version 2.0.0.

active_indicator_color is a ColorProperty and defaults to *None*.

inactive_indicator_color

The inactive indicator color in (r, g, b, a) or string format.

New in version 2.0.0.

inactive_indicator_color is a ColorProperty and defaults to *None*.

add_widget(*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_release(**args*) → None

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

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerItemLeadingIcon**(**args*, ***kwargs*)

Implements the leading icon for the menu list item.

For more information, see in the *MDListItemLeadingIcon* and *BaseNavigationDrawerItem* classes documentation.

New in version 2.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerItemText**(**args*, ***kwargs*)

Implements the text for the menu list item.

For more information, see in the *MDListItemSupportingText* and *BaseNavigationDrawerItem* classes documentation.

New in version 2.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerItemTrailingText**(**args*, ***kwargs*)

Implements the supporting text for the menu list item.

For more information, see in the *MDListItemTrailingSupportingText* and *BaseNavigationDrawerItem* classes documentation.

New in version 2.0.0.

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawerMenu**(**args*, ***kwargs*)

Implements a scrollable list for menu items of the *MDNavigationDrawer* class.

For more information, see in the *MDScrollView* class documentation.

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(*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)
```

update_items_color(*item:* MDNavigationDrawerItem) → None

class kivymd.uix.navigationdrawer.navigationdrawer.**MDNavigationDrawer**(**args*, ***kwargs*)

Navigation drawer class.

For more information, see in the *MDCard* class documentation.

Events

New in version 2.0.0.

***on_open*:**

Fired when the navigation drawer is opened.

***on_close*:**

Fired when the navigation drawer is closed.

drawer_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.

Changed in version 2.0.0: Rename from *type* to *drawer_type*.

drawer_type is a OptionProperty and defaults to *'modal'*.

anchor

Anchoring screen edge for drawer. Set it to *'right'* for right-to-left languages. Available options are: *'left'*, *'right'*.

anchor is a OptionProperty and defaults to *'left'*.

scrim_color

Color for scrim in (r, g, b, a) or string format. Alpha channel will be multiplied with `_scrim_alpha`. Set fourth channel to 0 if you want to disable scrim.

scrim_color is a ColorProperty and defaults to *[0, 0, 0, 0.5]*.

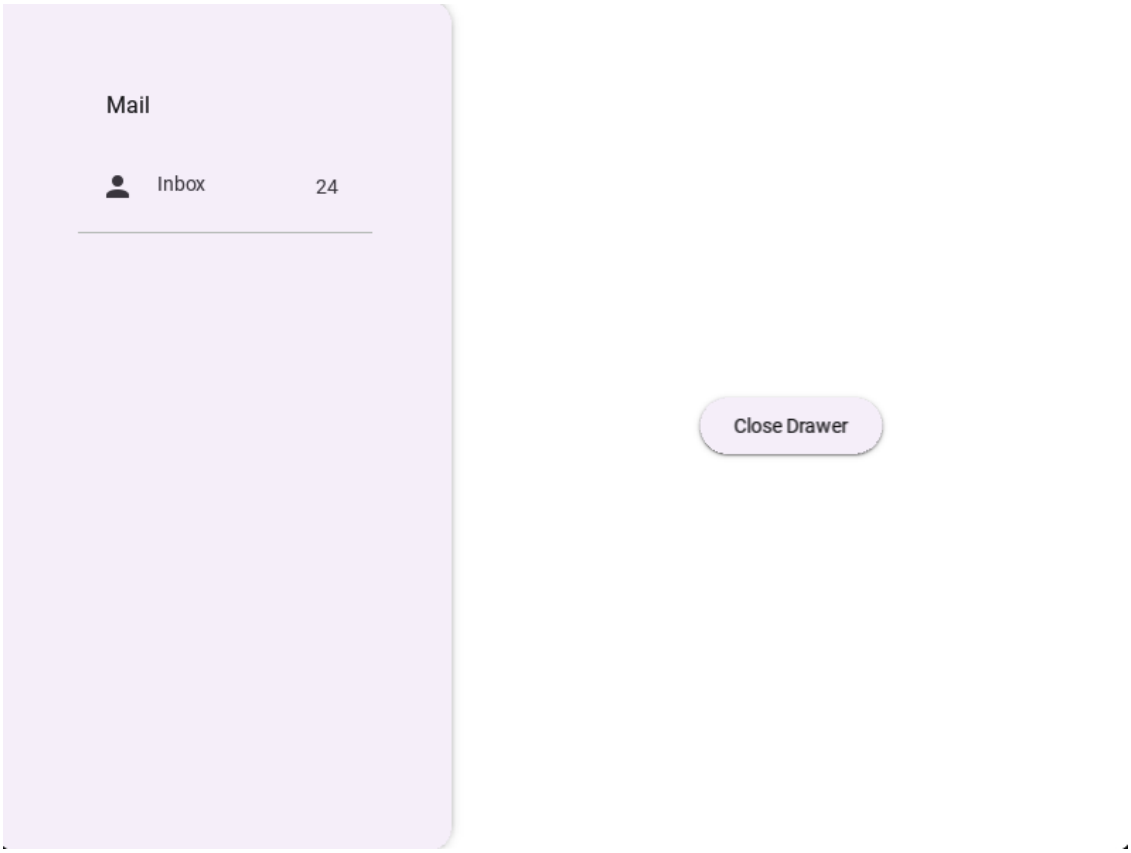
padding

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: "56dp"
```



padding is a VariableListProperty and defaults to '[dp(16), dp(16), dp(12), dp(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*.

background_color

The drawer background color in (r, g, b, a) or string format.

New in version 2.0.0.

background_color is a ColorProperty and defaults to *None*.

theme_elevation_level = 'Custom'

Drawer elevation level scheme name.

New in version 2.0.0.

Available options are: *'Primary'*, *'Custom'*.

theme_elevation_level is an OptionProperty and defaults to *'Custom'*.

elevation_level = 1

Drawer elevation level (values from 0 to 5)

New in version 2.2.0.

elevation_level is an BoundedNumericProperty and defaults to *2*.

set_properties_widget() → None

Fired *on_release/on_press/on_enter/on_leave* events.

set_state(*new_state='toggle'*, *animation=True*) → None

Change state of the side panel. New_state can be one of *"toggle"*, *"open"* or *"close"*.

`update_status`(**args*) → None

`get_dist_from_side`(*x: float*) → float

`on_touch_down`(*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`(*touch*)

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

See *on_touch_down()* for more information.

`on_touch_up`(*touch*)

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

See *on_touch_down()* for more information.

`on_radius`(*instance_navigation_drawer*, *radius_value: list*) → None

Fired when the `radius` value changes.

`on_drawer_type`(*instance_navigation_drawer*, *drawer_type: str*) → None

Fired when the *drawer_type* value changes.

`on_open`(**args*) → None

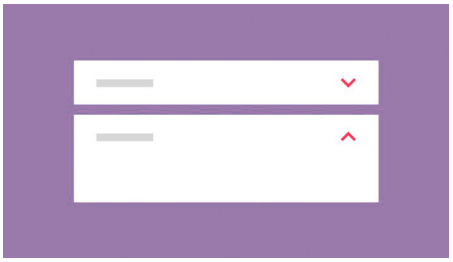
Fired when the navigation drawer is opened.

`on_close`(**args*) → None

Fired when the navigation drawer is closed.

2.3.19 ExpansionPanel

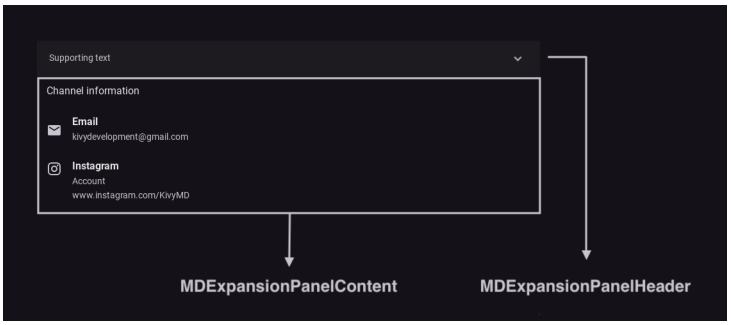
Expansion panels contain creation flows and allow lightweight editing of an element.



Usage

```
MDExpansionPanel:

    MDExpansionPanelHeader:

        # Content of header.
        [...]

    MDExpansionPanelContent:

        # Content of panel.
        [...]
```

Anatomy



Example

```
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import RotateBehavior
from kivymd.uix.expansionpanel import MDExpansionPanel
from kivymd.uix.list import MDListItemTrailingIcon

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDExpansionPanel:
        id: panel
        pos_hint: {"center_x": .5, "center_y": .5}

        MDExpansionPanelHeader:

            MDListItem:
                theme_bg_color: "Custom"
```

(continues on next page)

(continued from previous page)

```
                md_bg_color: self.theme_cls.surfaceContainerLowColor
                ripple_effect: False

                MDListItemSupportingText:
                    text: "Supporting text"

                TrailingPressedIconButton:
                    id: chevron
                    icon: "chevron-right"
                    on_release: app.tap_expansion_chevron(panel, chevron)

        MDExpansionPanelContent:
            orientation: "vertical"
            padding: "12dp", 0, "12dp", 0

            MDLabel:
                text: "Channel information"
                adaptive_height: True
                padding_x: "16dp"
                padding_y: "12dp"

            MDListItem:

                MDListItemLeadingIcon:
                    icon: "email"

                MDListItemHeadlineText:
                    text: "Email"

                MDListItemSupportingText:
                    text: "kivydevelopment@gmail.com"

            MDListItem:

                MDListItemLeadingIcon:
                    icon: "instagram"

                MDListItemHeadlineText:
                    text: "Instagram"

                MDListItemSupportingText:
                    text: "Account"

                MDListItemTertiaryText:
                    text: "www.instagram.com/KivyMD"
'''


class TrailingPressedIconButton(
    ButtonBehavior, RotateBehavior, MDListItemTrailingIcon
):
    ...
```

(continues on next page)

(continued from previous page)

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

    def tap_expansion_chevron(
        self, panel: MDExpansionPanel, chevron: TrailingPressedIconButton
    ):
        panel.open() if not panel.is_open else panel.close()
        panel.set_chevron_down(
            chevron
        ) if not panel.is_open else panel.set_chevron_up(chevron)


Example().run()
```

Use with ScrollView

```
import asynckivy
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.metrics import dp
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import RotateBehavior
from kivymd.uix.expansionpanel import MDExpansionPanel
from kivymd.uix.list import MDListItemTrailingIcon

KV = '''
<ExpansionPanelItem>

    MDExpansionPanelHeader:

        MDListItem:
            theme_bg_color: "Custom"
            md_bg_color: self.theme_cls.surfaceContainerLowColor
            ripple_effect: False

            MDListItemSupportingText:
                text: "Supporting text"

            TrailingPressedIconButton:
                id: chevron
                icon: "chevron-right"
                on_release: app.tap_expansion_chevron(root, chevron)
```

(continues on next page)

(continued from previous page)

```
    MDExpansionPanelContent:
        orientation: "vertical"
        padding: "12dp", 0, "12dp", "12dp"
        md_bg_color: self.theme_cls.surfaceContainerLowestColor

        MDLabel:
            text: "Channel information"
            adaptive_height: True
            padding_x: "16dp"
            padding_y: "12dp"

        MDListItem:
            theme_bg_color: "Custom"
            md_bg_color: self.theme_cls.surfaceContainerLowestColor

            MDListItemLeadingIcon:
                icon: "email"

            MDListItemHeadlineText:
                text: "Email"

            MDListItemSupportingText:
                text: "kivydevelopment@gmail.com"

        MDListItem:
            theme_bg_color: "Custom"
            md_bg_color: self.theme_cls.surfaceContainerLowestColor

            MDListItemLeadingIcon:
                icon: "instagram"

            MDListItemHeadlineText:
                text: "Instagram"

            MDListItemSupportingText:
                text: "Account"

            MDListItemTertiaryText:
                text: "www.instagram.com/KivyMD"


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    ScrollView:
        size_hint_x: .5
        pos_hint: {"center_x": .5, "center_y": .5}

        MDList:
            id: container
'''
```

(continues on next page)

(continued from previous page)

```
class ExpansionPanelItem(MDExpansionPanel):
    ...


class TrailingPressedIconButton(
    ButtonBehavior, RotateBehavior, MDListItemTrailingIcon
):
    ...


class Example(MDApp):
    def on_start(self):
        async def set_panel_list():
            for i in range(12):
                await asynckivy.sleep(0)
                self.root.ids.container.add_widget(ExpansionPanelItem())

        asynckivy.start(set_panel_list())

    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

    def tap_expansion_chevron(
        self, panel: MDExpansionPanel, chevron: TrailingPressedIconButton
    ):
        Animation(
            padding=[0, dp(12), 0, dp(12)]
            if not panel.is_open
            else [0, 0, 0, 0],
            d=0.2,
        ).start(panel)
        panel.open() if not panel.is_open else panel.close()
        panel.set_chevron_down(
            chevron
        ) if not panel.is_open else panel.set_chevron_up(chevron)


Example().run()
```

API break

1.2.0 version

```
MDExpansionPanel(
    icon="icon.png",
    content=Content(),  # content of panel
    panel_cls=MDExpansionPanelThreeLine(  # content of header
        text="Text",
        secondary_text="Secondary text",
        tertiary_text="Tertiary text",
    )
)
```

2.0.0 version

```
MDExpansionPanel:

    MDExpansionPanelHeader:

        # Content of header.
        [...]

    MDExpansionPanelContent:

        # Content of panel.
        [...]
```

API - kivymd.uix.expansionpanel.expansionpanel

class kivymd.uix.expansionpanel.expansionpanel.**MDExpansionPanelContent**(**args*, ***kwargs*)

Implements a container for panel content.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and BoxLayout classes documentation.

class kivymd.uix.expansionpanel.expansionpanel.**MDExpansionPanelHeader**(**args*, ***kwargs*)

Implements a container for the content of the panel header.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

class kivymd.uix.expansionpanel.expansionpanel.**MDExpansionPanel**(***kwargs*)

Expansion panel class.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

Events

on_open
Fired when a panel is opened.

on_close
Fired when a panel is closed.

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*.

is_open
The panel is open or closed.

New in version 2.0.0.

is_open is a `BooleanProperty` and defaults to *False*.

on_open(**args*) → None
Fired when a panel is opened.

on_close(**args*) → None
Fired when a panel is closed.

set_chevron_down(*instance*) → None
Sets the chevron down.

set_chevron_up(*instance*) → None
Sets the chevron up.

close(**args*) → None
Method closes the panel.

Changed in version 2.0.0: Rename from *close_panel* to *close* method.

open(**args*) → None
Method opens a panel.

Changed in version 2.0.0: Rename from *open_panel* to *open* method.

add_widget(*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.

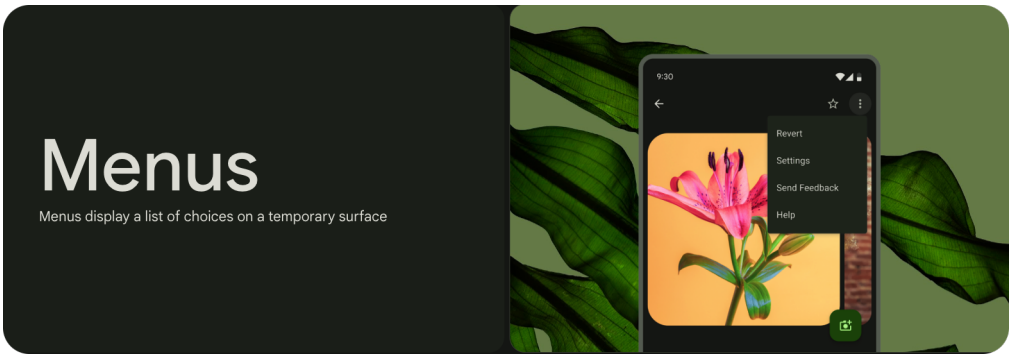
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.20 Menu

See also:

Material Design spec, Menus

Menus display a list of choices on temporary surfaces.

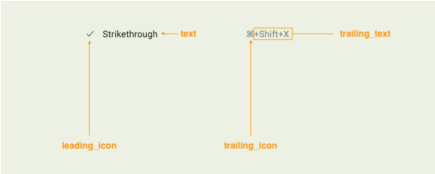


- Menus should be easy to open, close, and interact with
- Menu content should be suited to user needs
- Menu items should be easy to scan

Usage

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

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 menu_open(self):
        menu_items = [
            {
                "text": f"Item {i}",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            } for i in range(5)
        ]
        MDDropdownMenu(
            caller=self.root.ids.button, items=menu_items
        ).open()

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

    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)


Test().run()
```

Anatomy

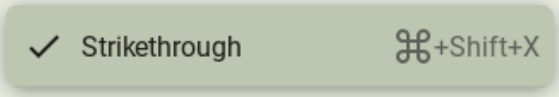


You can combine the following parameters:

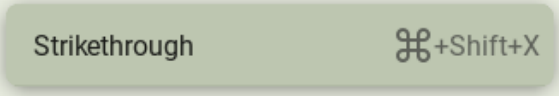
- leading_icon
- text
- trailing_icon
- trailing_text

. . . to create the necessary types of menu items:

```
menu_items = [
    {
        "text": "Strikethrough",
        "leading_icon": "check",
        "trailing_icon": "apple-keyboard-command",
        "trailing_text": "+Shift+X",
    }
]
```




```
menu_items = [
    {
        "text": "Strikethrough",
        "trailing_icon": "apple-keyboard-command",
        "trailing_text": "+Shift+X",
    }
]
```

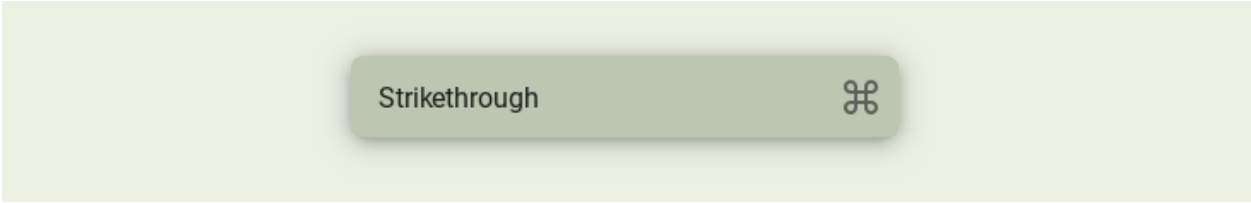



```
menu_items = [
    {
```

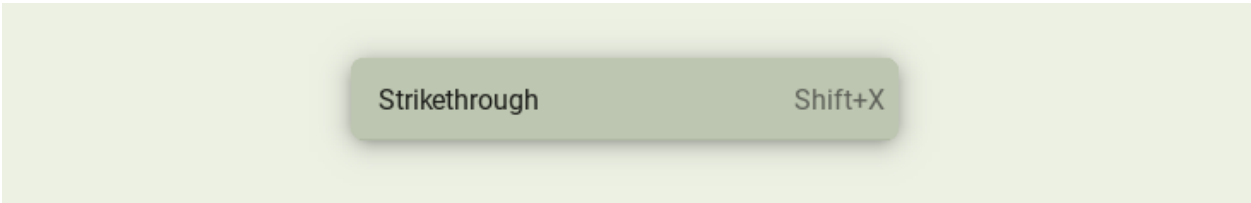
(continues on next page)

(continued from previous page)

```
        "text": "Strikethrough",
        "trailing_icon": "apple-keyboard-command",
    }
]
```

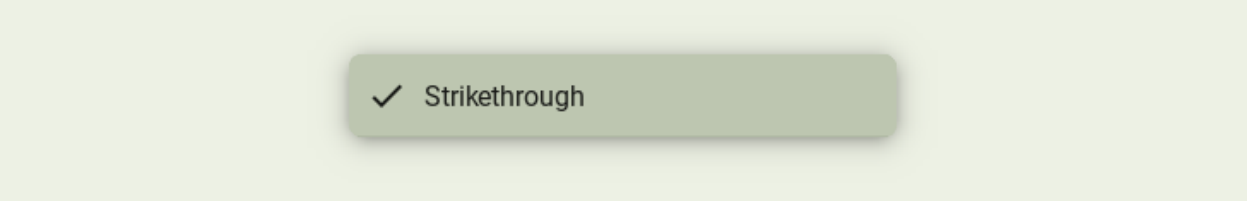



```
menu_items = [
    {
        "text": "Strikethrough",
        "trailing_text": "Shift+X",
    }
]
```

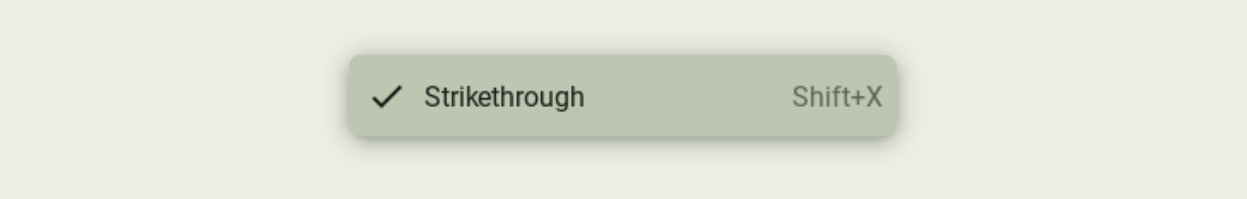



```
menu_items = [
    {
        "text": "Strikethrough",
        "leading_icon": "check",
        "trailing_icon": "apple-keyboard-command",
    }
]
```

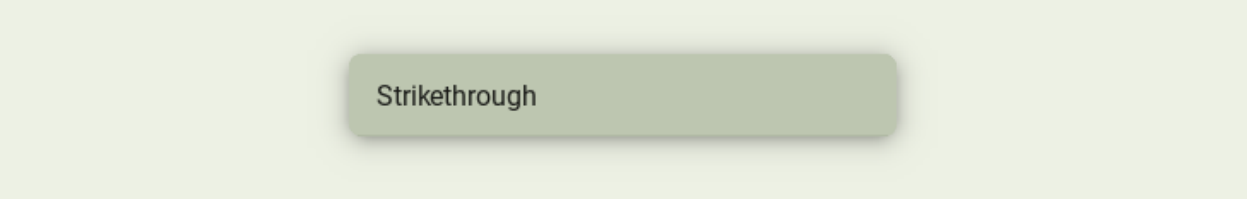

```
menu_items = [
    {
        "text": "Strikethrough",
        "leading_icon": "check",
    }
]
```




```
menu_items = [
    {
        "text": "Strikethrough",
        "leading_icon": "check",
        "trailing_text": "Shift+X",
    }
]
```




```
menu_items = [
    {
        "text": "Strikethrough",
    }
]
```



You can use the following parameters to customize the menu items:

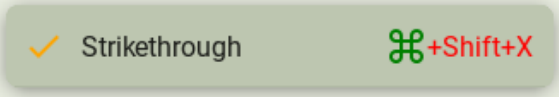
- text_color
- leading_icon_color
- trailing_icon_color
- trailing_text_color

```
menu_items = [
    {
        "text": "Strikethrough",
        "leading_icon": "check",
        "trailing_icon": "apple-keyboard-command",
        "trailing_text": "+Shift+X",
        "leading_icon_color": "orange",
        "trailing_icon_color": "green",
```

(continues on next page)

(continued from previous page)

```
        "trailing_text_color": "red",
    }
]
```



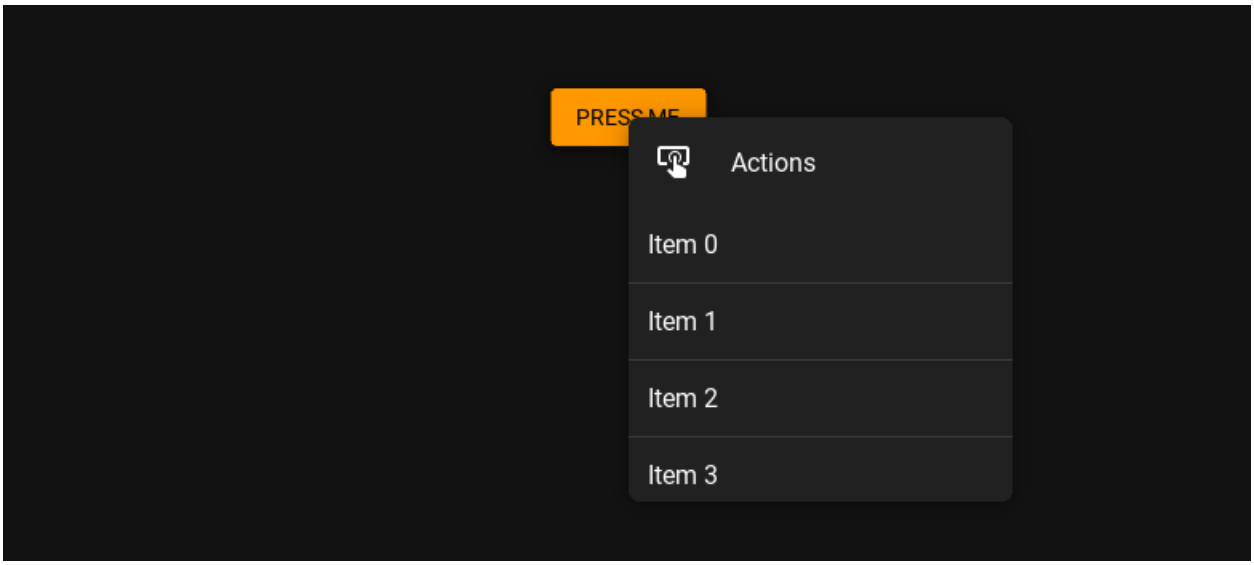
Header

```
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>
    spacing: "12dp"
    padding: "4dp"
    adaptive_height: True

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

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


MDScreen:

    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.'''
```

(continues on next page)

(continued from previous page)

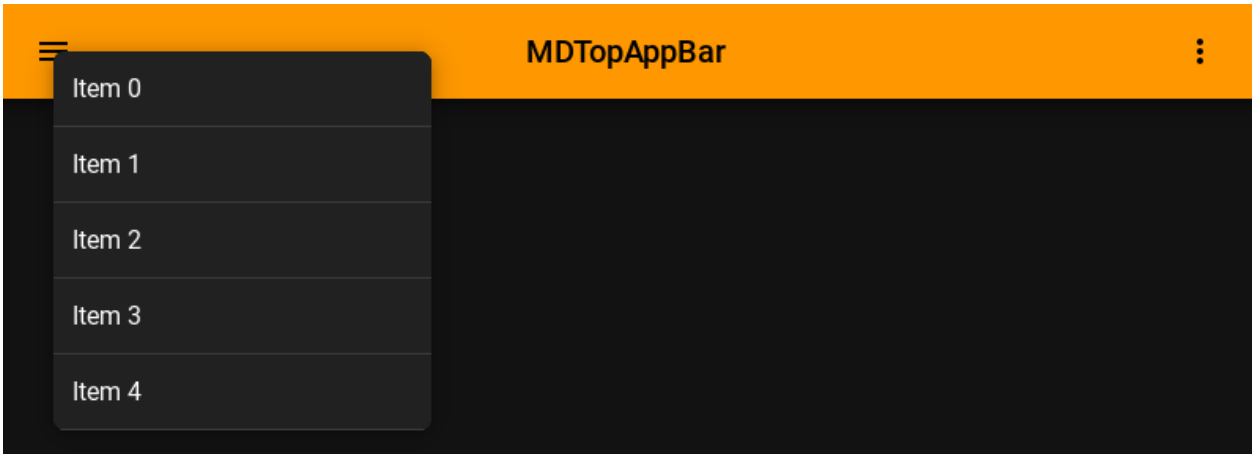
```
class Test(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = Builder.load_string(KV)
        menu_items = [
            {
                "text": f"Item {i}",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            }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

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

    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"
        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)*. This example uses drop down menus for both the righthand and lefthand menus.

```
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)]]

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


class Test(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"
        menu_items = [
            {
                "text": f"Item {i}",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            } for i in range(5)
        ]
        self.menu = MDDropdownMenu(items=menu_items)
        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:

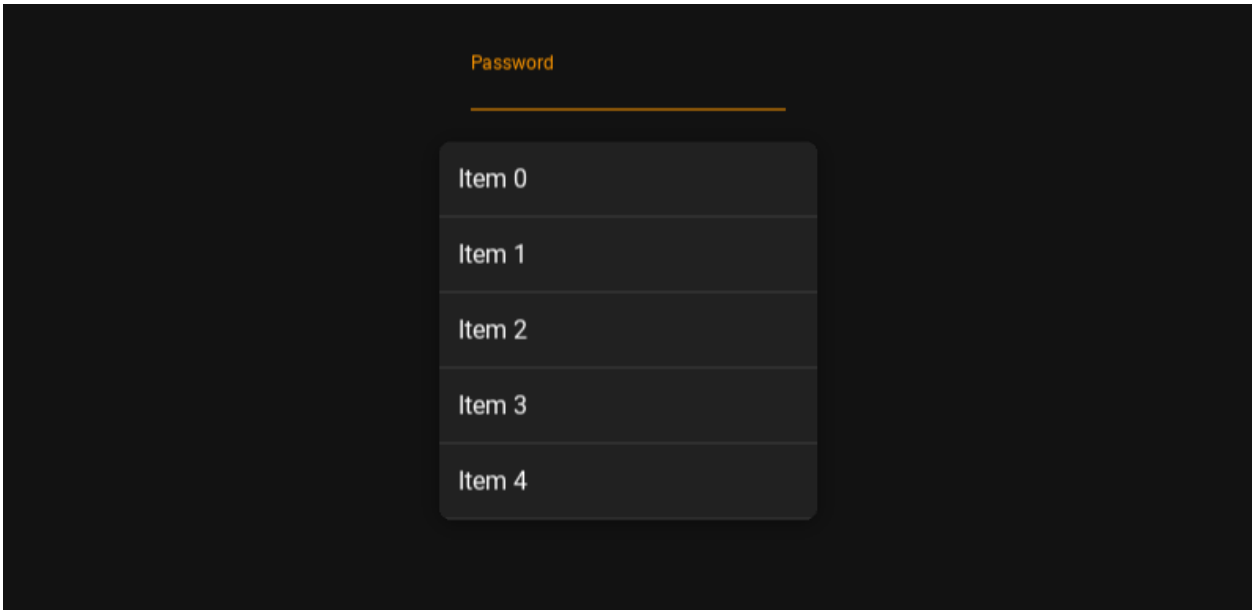
position

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

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

KV = '''
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 Test(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = Builder.load_string(KV)
        menu_items = [
            {
                "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,
```

(continues on next page)

(continued from previous page)

```
            items=menu_items,
            position="bottom",
        )

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

    def build(self):
        self.theme_cls.primary_palette = "Orange"
        self.theme_cls.theme_style = "Dark"
        return self.screen


Test().run()
```



Center position

```
from kivy.lang import Builder

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

KV = '''
MDScreen
    md_bg_color: self.theme_cls.backgroundColor

    MDDropDownItem:
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.open_drop_item_menu(self)
```

(continues on next page)

(continued from previous page)

```
        MDDropDownItemText:
            id: drop_text
            text: "Item"
'''


class Example(MDApp, CommonApp):
    drop_item_menu: MDDropdownMenu = None

    def open_drop_item_menu(self, item):
        menu_items = [
            {
                "text": f"{i}",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            } for i in range(5)
        ]
        if not self.drop_item_menu:
            self.drop_item_menu = MDDropdownMenu(
                caller=item, items=menu_items, position="center"
            )
            self.drop_item_menu.open()

    def menu_callback(self, text_item):
        self.root.ids.drop_text.text = text_item
        self.drop_item_menu.dismiss()

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

API break

1.1.1 version

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

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

KV = '''
<RightContentCls>
    disabled: True
    adaptive_size: True
```

(continues on next page)

(continued from previous page)

```
    pos_hint: {"center_y": .5}

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

    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 = [
            {
```

(continues on next page)

(continued from previous page)

```
                "text": f"Item {i}",
                "right_text": "+Shift+X",
                "right_icon": "apple-keyboard-command",
                "left_icon": "web",
                "viewclass": "Item",
                "height": dp(54),
                "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,
            bg_color="#bdc6b0",
            width_mult=4,
        )

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

    def build(self):
        return self.screen


Test().run()
```

1.2.0 version

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

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 = [
            {
```

(continues on next page)

(continued from previous page)

```
                "text": f"Item {i}",
                "leading_icon": "web",
                "trailing_icon": "apple-keyboard-command",
                "trailing_text": "+Shift+X",
                "trailing_icon_color": "grey",
                "trailing_text_color": "grey",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            } for i in range(5)
        ]
        self.menu = MDDropdownMenu(
            md_bg_color="#bdc6b0",
            caller=self.screen.ids.button,
            items=menu_items,
        )

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

    def build(self):
        return self.screen


Test().run()
```

API - kivymd.uix.menu.menu

class kivymd.uix.menu.menu.**BaseDropdownItem**(***kwargs*)

Base class for menu items.

New in version 1.2.0.

For more information, see in the *RectangularRippleBehavior* and *MDBoxLayout* classes.

text

The text of the menu item.

text is a StringProperty and defaults to ‘’.

leading_icon

The leading icon of the menu item.

leading_icon is a StringProperty and defaults to ‘’.

trailing_icon

The trailing icon of the menu item.

trailing_icon is a StringProperty and defaults to ‘’.

trailing_text

The trailing text of the menu item.

trailing_text is a StringProperty and defaults to ‘’.

text_color

The color of the text in (r, g, b, a) or string format for the text of the menu item.

text_color is a ColorProperty and defaults to *None*.

leading_icon_color

The color of the text in (r, g, b, a) or string format for the leading icon of the menu item.

leading_icon_color is a ColorProperty and defaults to *None*.

trailing_icon_color

The color of the text in (r, g, b, a) or string format for the trailing icon of the menu item.

leading_icon_color is a ColorProperty and defaults to *None*.

trailing_text_color

The color of the text in (r, g, b, a) or string format for the trailing text of the menu item.

leading_icon_color is a ColorProperty and defaults to *None*.

divider

Divider mode. Available options are: *'Full'*, *None* and default to *'Full'*.

divider is a OptionProperty and defaults to *'Full'*.

divider_color

Divider color in (r, g, b, a) or string format.

divider_color is a ColorProperty and defaults to *None*.

class kivymd.uix.menu.menu.**MDDropdownTextItem**(***kwargs*)

Implements a menu item with text without leading and trailing icons.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownLeadingIconItem**(***kwargs*)

Implements a menu item with text, leading icon and without trailing icon.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownTrailingIconItem**(***kwargs*)

Implements a menu item with text, without leading icon and with trailing icon.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownTrailingIconTextItem**(***kwargs*)

Implements a menu item with text, without leading icon, with trailing icon and with trailing text.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownTrailingTextItem**(***kwargs*)

Implements a menu item with text, without leading icon, without trailing icon and with trailing text.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownLeadingIconTrailingTextItem**(***kwargs*)

Implements a menu item with text, leading icon and with trailing text.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownLeadingTrailingIconTextItem**(***kwargs*)

Implements a menu item with text, with leading icon, with trailing icon and with trailing text.

New in version 1.2.0.

For more information, see in the *BaseDropdownItem* class.

class kivymd.uix.menu.menu.**MDDropdownMenu**(***kwargs*)

Dropdown menu class.

For more information, see in the *MotionDropDownMenuBehavior* and *StencilBehavior* and *MDCard* classes documentation.

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

List of dictionaries with properties for menu 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.

Deprecated since version 1.2.0: Use *width* instead.

```
self.menu = MDDropdownMenu(
    width=dp(240),
    ...,
)
```

width_mult is a NumericProperty and defaults to *1*.

min_height

max_height

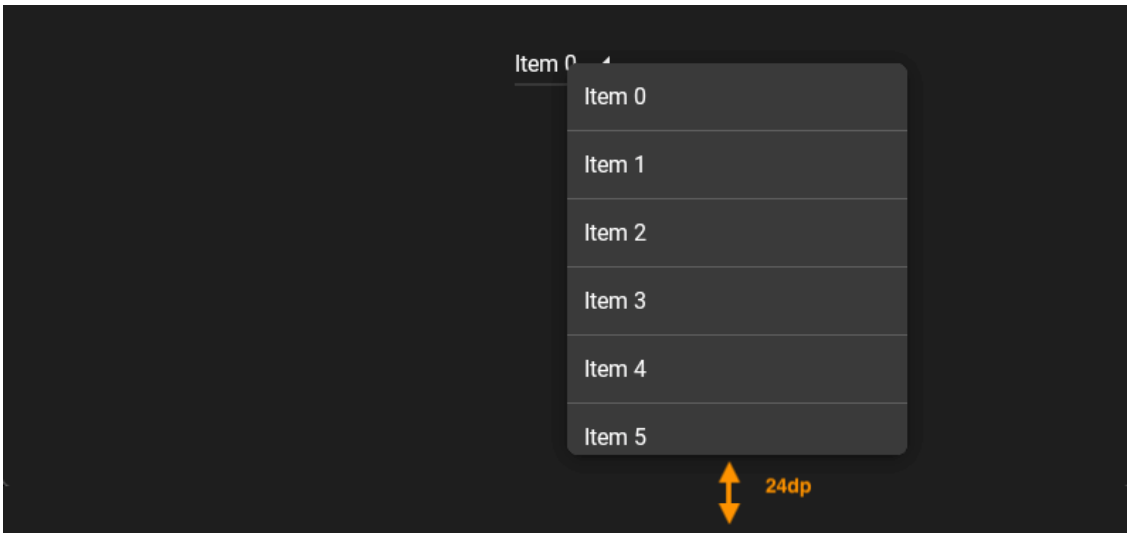
The menu will grow no bigger than this number. Set to 0 for no limit.

max_height is a NumericProperty and defaults to *0*.

border_margin

Margin between Window border and menu.

```
self.menu = MDDropdownMenu(
    border_margin=dp(24),
    ...,
)
```

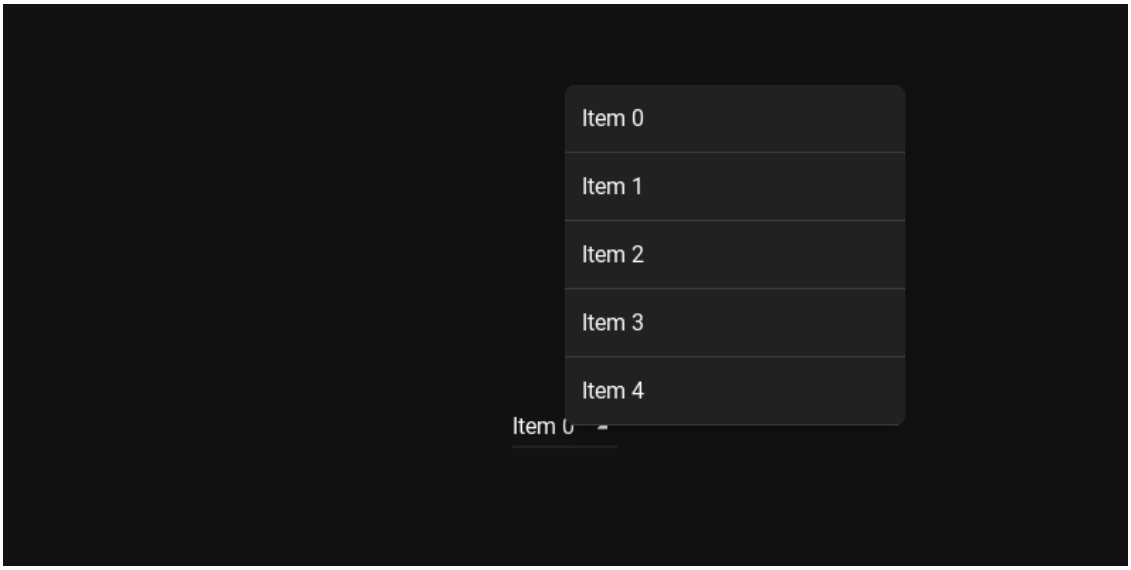


border_margin is a NumericProperty and defaults to *4dp*.

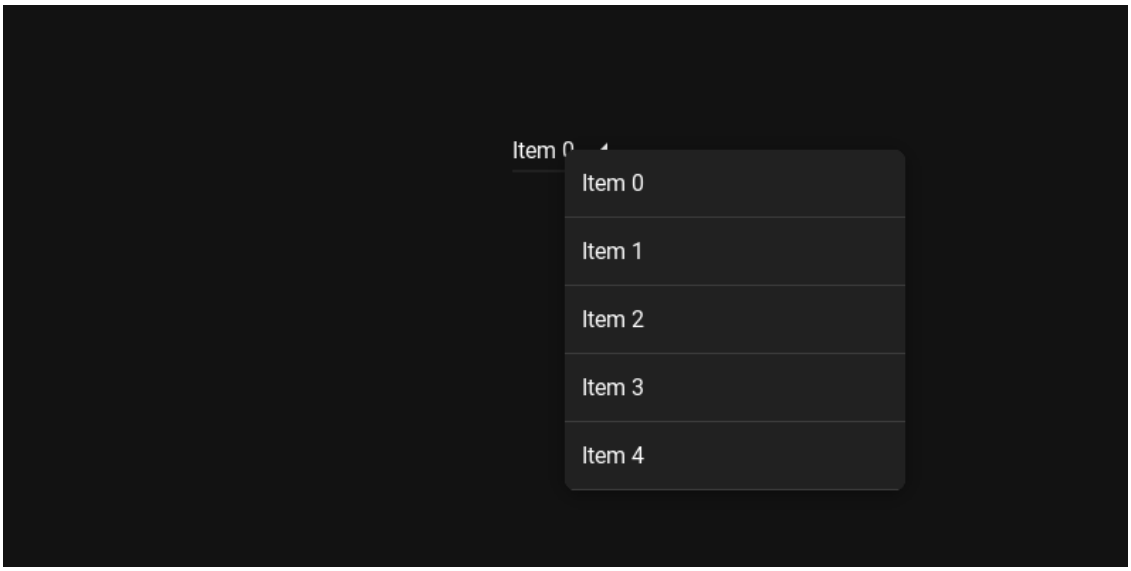
ver_growth

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",
    ...,
)
```

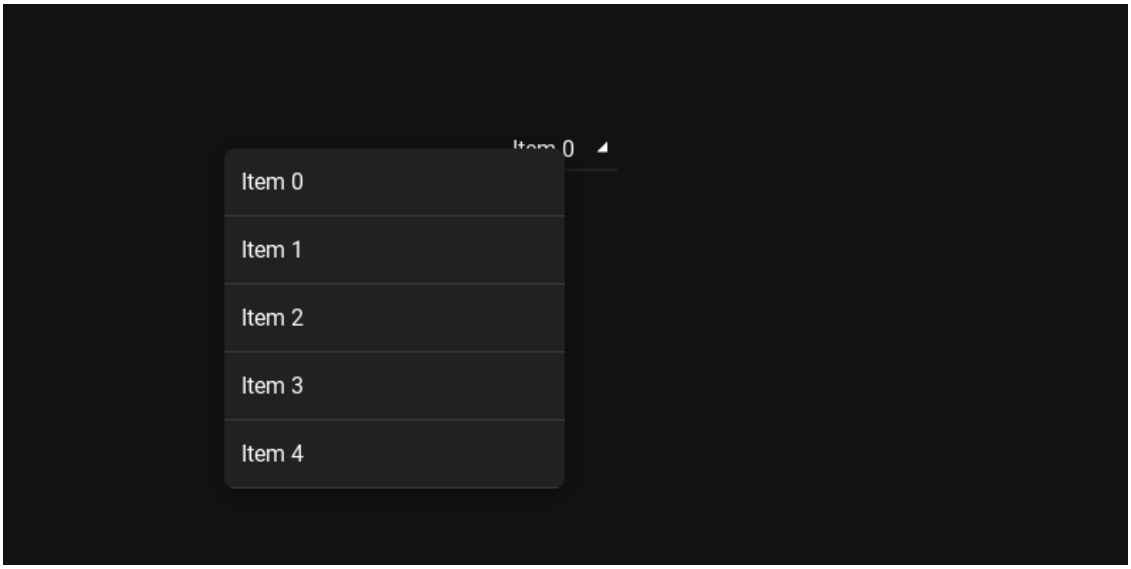


ver_growth is a OptionProperty and defaults to *None*.

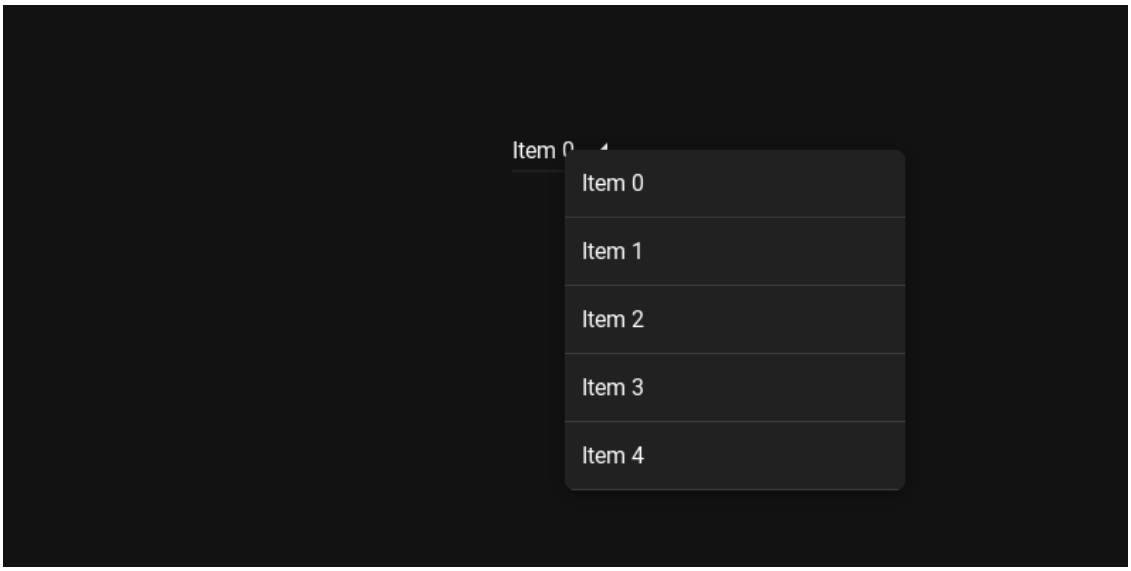
hor_growth

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 in (r, g, b, a) or string format of the background of the menu.

Deprecated since version 1.2.0: Use *md_bg_color* instead.

background_color is a ColorProperty and defaults to *None*.

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'*, *'top'*, *'center'*, *'bottom'*.

See *Position* for more information.

position is a OptionProperty and defaults to *'auto'*.

radius

Menu radius.

radius is a VariableListProperty and defaults to *'[dp(7)]'*.

adjust_width() → None

Adjust the width of the menu if the width of the menu goes beyond the boundaries of the parent window from starting point.

check_ver_growth() → None

Checks whether the height of the lower/upper borders of the menu exceeds the limits borders of the parent window.

check_hor_growth() → None

Checks whether the width of the left/right menu borders exceeds the boundaries of the parent window.

get_target_pos() → [float, float]

set_target_height() → None

Set the target height of the menu depending on the size of each item.

set_menu_properties(**args*) → None

Sets the size and position for the menu window.

set_menu_pos(**args*) → None

adjust_position() → str

Return value 'auto' for the menu position if the menu position is out of screen.

open() → None

Animate the opening of a menu window.

on_items(*instance*, *value: list*) → None

The method sets the class that will be used to create the menu item.

on_header_cls(*instance_dropdown_menu*, *instance_user_menu_header*) → None

Called when a value is set to the *header_cls* parameter.

on_touch_down(*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(*touch*)

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

See *on_touch_down()* for more information.

on_touch_up(*touch*)

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

See *on_touch_down()* for more information.

dismiss(**args*) → None

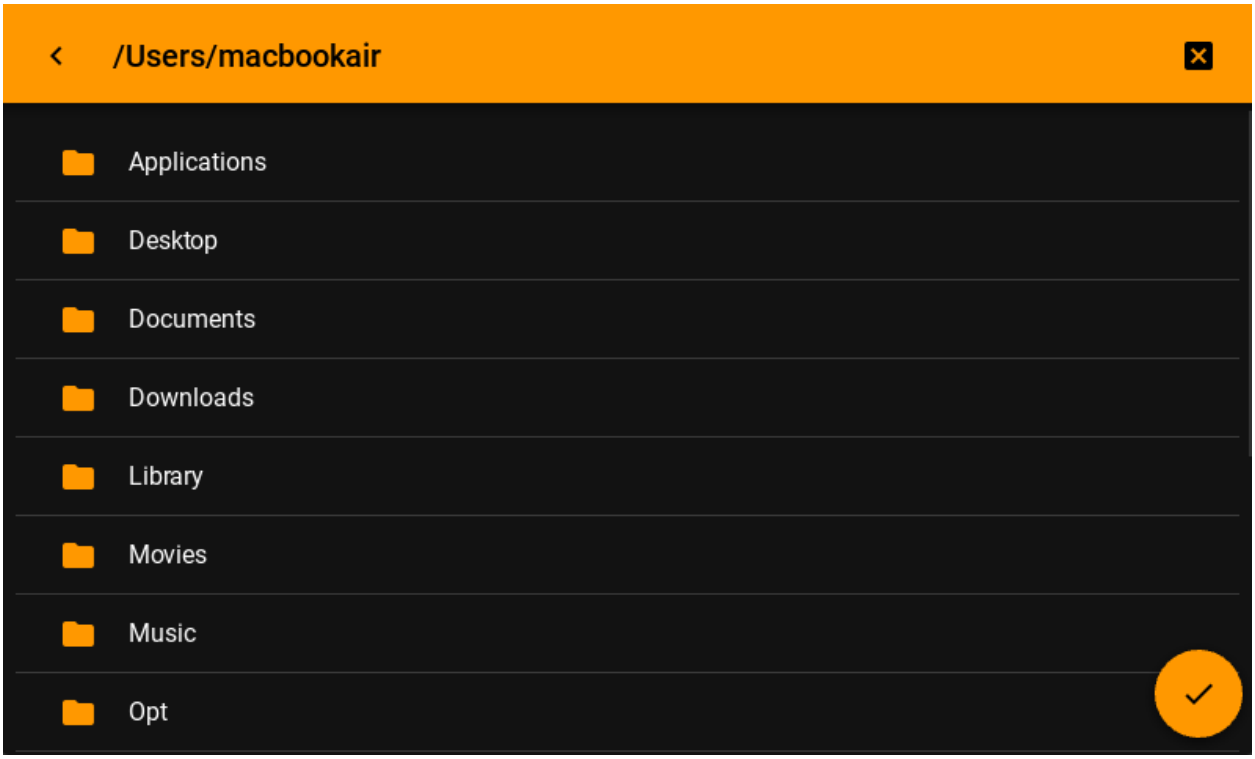
Closes the menu.

2.3.21 FileManager

A simple manager for selecting directories and files.

Usage

```
path = os.path.expanduser("~")  # 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.

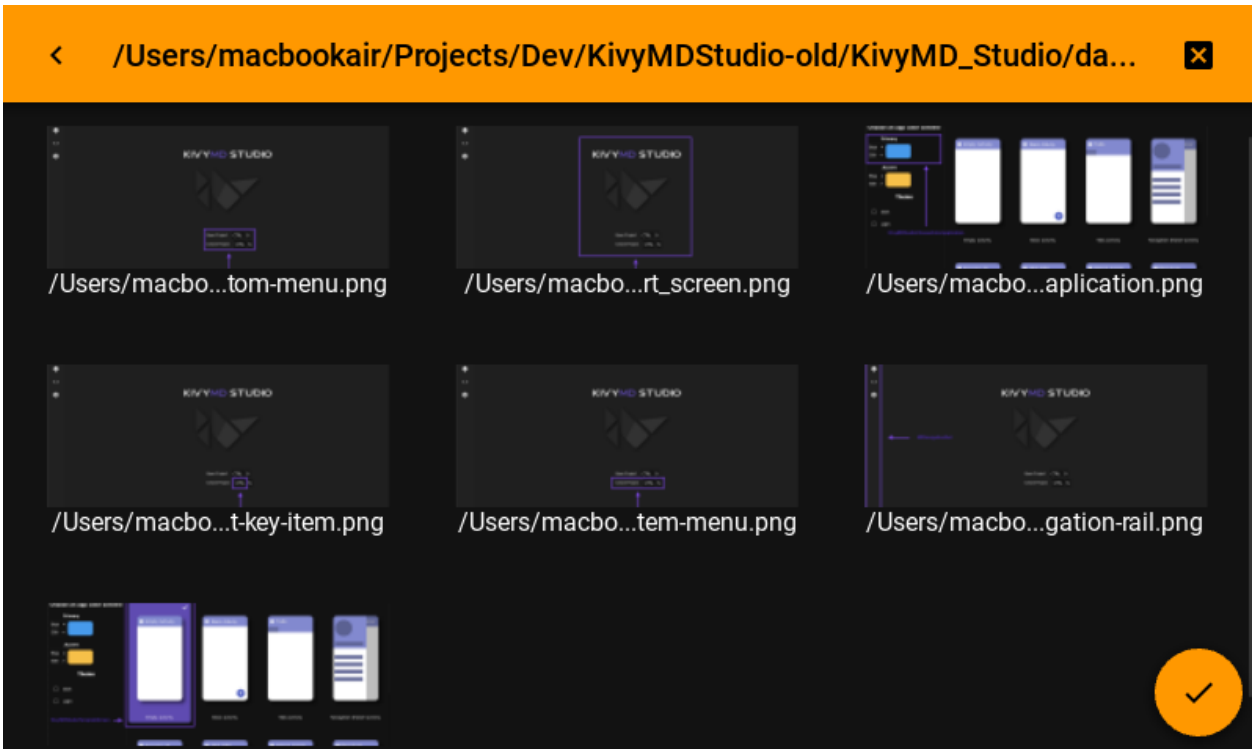
Or with `preview` mode:

```
file_manager = MDFileManager(
    exit_manager=self.exit_manager,
    select_path=self.select_path,
```

(continues on next page)

(continued from previous page)

```
    preview=True,
)
```



Warning: The *preview* mode is intended only for viewing images and will not display other types of files.

Example

```
import os

from kivy.core.window import Window
from kivy.lang import Builder
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.filemanager import MDFileManager
from kivymd.uix.snackbar import MDSnackbar, MDSnackbarText

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.file_manager_open()
```

(continues on next page)

(continued from previous page)

```
        MDButtonText:
            text: "Open manager"
'''


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        Window.bind(on_keyboard=self.events)
        self.manager_open = False
        self.file_manager = MDFileManager(
            exit_manager=self.exit_manager, select_path=self.select_path
        )

    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)

    def file_manager_open(self):
        self.file_manager.show(
            os.path.expanduser("~"))  # output manager to the screen
        self.manager_open = True

    def select_path(self, path: str):
        '''
        It will be called when you click on the file name
        or the catalog selection button.

        :param path: path to the selected directory or file;
        '''

        self.exit_manager()
        MDSnackbar(
            MDSnackbarText(
                text=path,
            ),
            y=dp(24),
            pos_hint={"center_x": 0.5},
            size_hint_x=0.8,
        ).open()

    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:
```

(continues on next page)

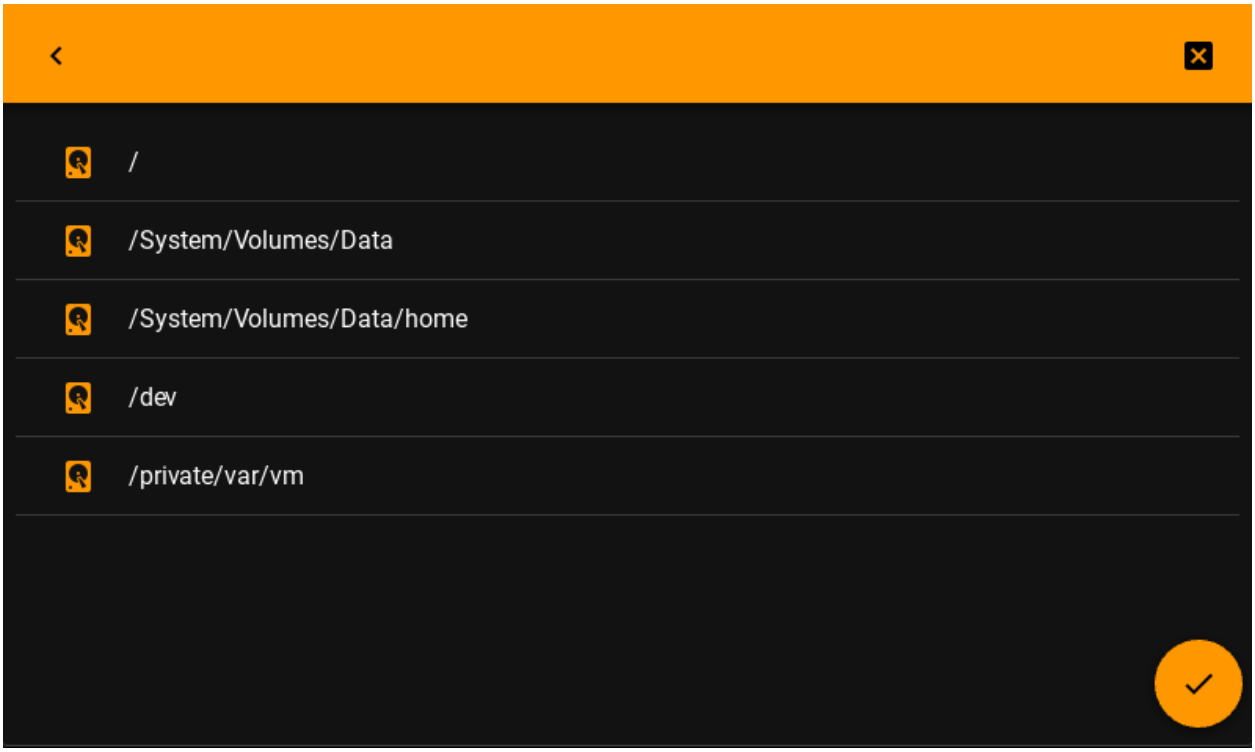
(continued from previous page)

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

Implements a modal dialog with a file manager.

For more information, see in the *ThemableBehavior* and RelativeLayout classes documentation.

Events

***on_pre_open*:**
Called before the MDFileManager is opened.

***on_open*:**
Called when the MDFileManager is opened.

***on_pre_dismiss*:**
Called before the MDFileManager is closed.

on_dismiss:
Called when the MDFileManager is closed.

`icon`

Icon that will be used on the directory selection button.

Deprecated since version 1.1.0: Use *icon_selection_button* instead.

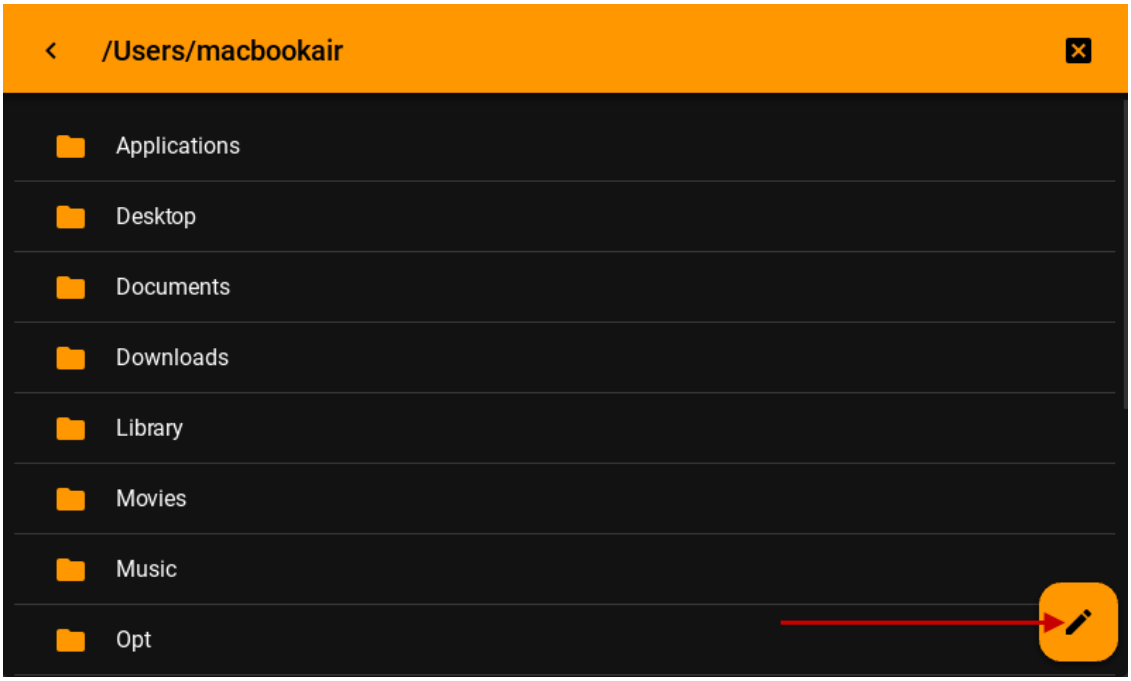
icon is an StringProperty and defaults to *check*.

`icon_selection_button`

Icon that will be used on the directory selection button.

New in version 1.1.0.

```
MDFileManager(
    ...
    icon_selection_button="pencil",
)
```



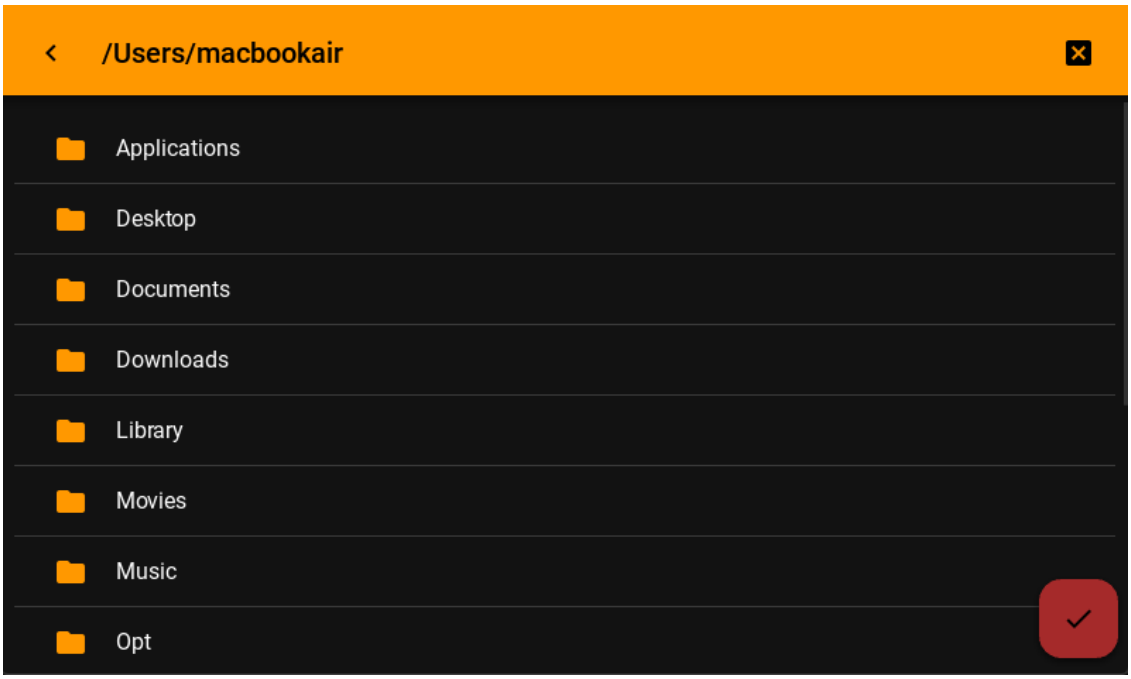
icon_selection_button is an StringProperty and defaults to *check*.

`background_color_selection_button`

Background color in (r, g, b, a) or string format of the current directory/path selection button.

New in version 1.1.0.

```
MDFileManager(
    ...
    background_color_selection_button="brown",
)
```



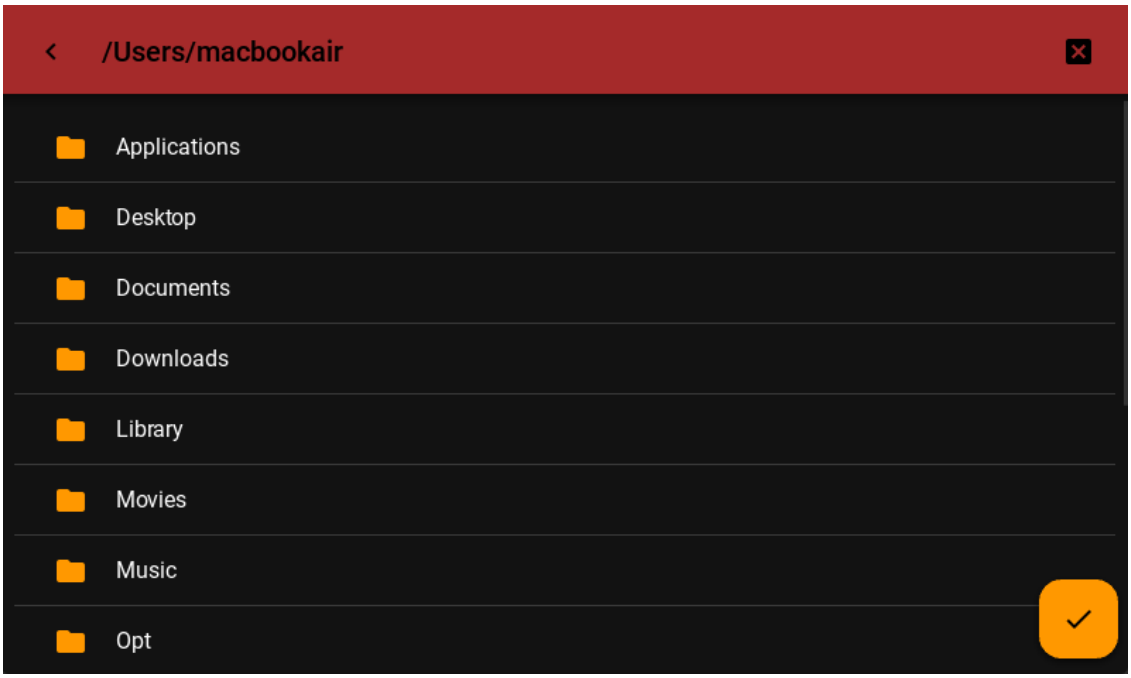
background_color_selection_button is an ColorProperty and defaults to *None*.

background_color_toolbar

Background color in (r, g, b, a) or string format of the file manager toolbar.

New in version 1.1.0.

```
MDFileManager(
    ...
    background_color_toolbar="brown",
)
```

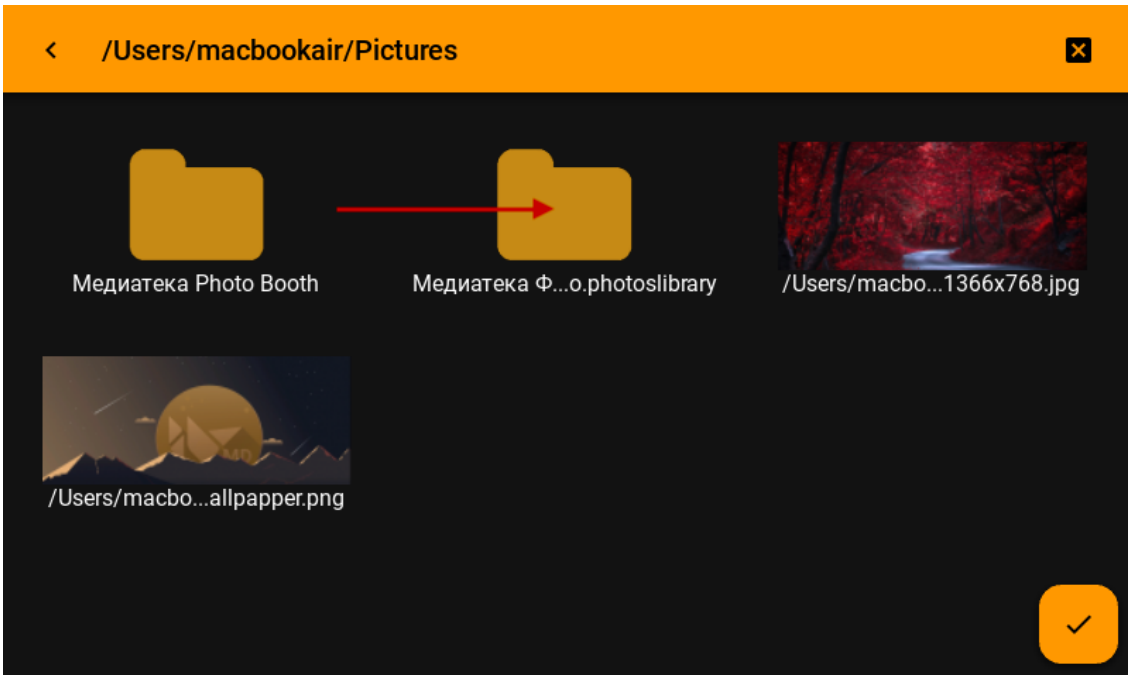


background_color_toolbar is an ColorProperty and defaults to *None*.

icon_folder

Icon that will be used for folder icons when using `preview = True`.

```
MDFileManager(
    ...
    preview=True,
    icon_folder="path/to/icon.png",
)
```



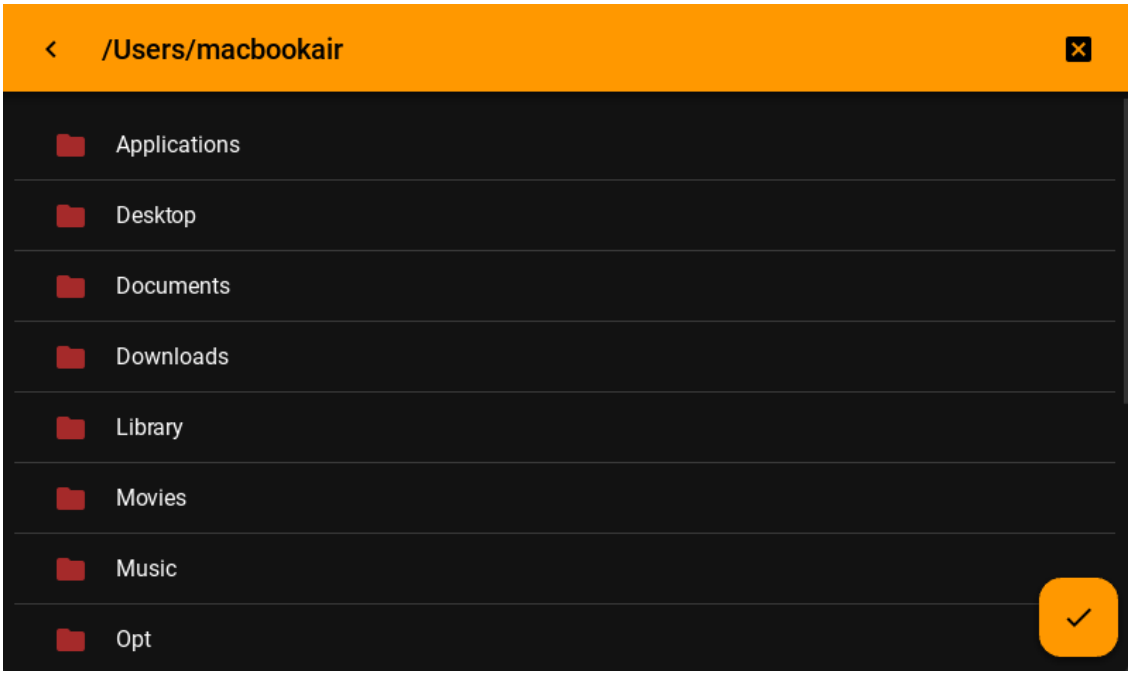
icon is an StringProperty and defaults to *check*.

icon_color

Color in (r, g, b, a) or string format of the folder icon when the *preview* property is set to False.

New in version 1.1.0.

```
MDFileManager(
    ...
    preview=False,
    icon_color="brown",
)
```



icon_color is an ColorProperty and defaults to *None*.

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 *os.path.expanduser("~")*.

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 *[]*.

selection_button

The instance of the directory/path selection button.

New in version 1.1.0.

selection_button is a read-only ObjectProperty and defaults to *None*.

show_disks() → None

show(*path: str*) → None

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(*path: str*) → str

get_content() → Tuple[List[str], List[str]] | Tuple[None, None]

Returns a list of the type [[Folder List], [file list]].

close() → None

Closes the file manager window.

select_dir_or_file(*path: str*, *widget: MDFileManagerItemPreview | MDFileManagerItem*) → None

Called by tap on the name of the directory or file.

back() → None

Returning to the branch down in the directory tree.

select_directory_on_press_button(**args*) → None

Called when a click on a floating button.

on_icon(*instance_file_manager*, *icon_name: str*) → None

Called when the *icon* property is changed.

on_background_color_toolbar(*instance_file_manager*, *color: str | list*) → None

Called when the *background_color_toolbar* property is changed.

on_pre_open(**args*) → None

Default pre-open event handler.

New in version 1.1.0.

on_open(**args*) → None

Default open event handler.

New in version 1.1.0.

on_pre_dismiss(**args*) → None

Default pre-dismiss event handler.

New in version 1.1.0.

on_dismiss(**args*) → None

Default dismiss event handler.

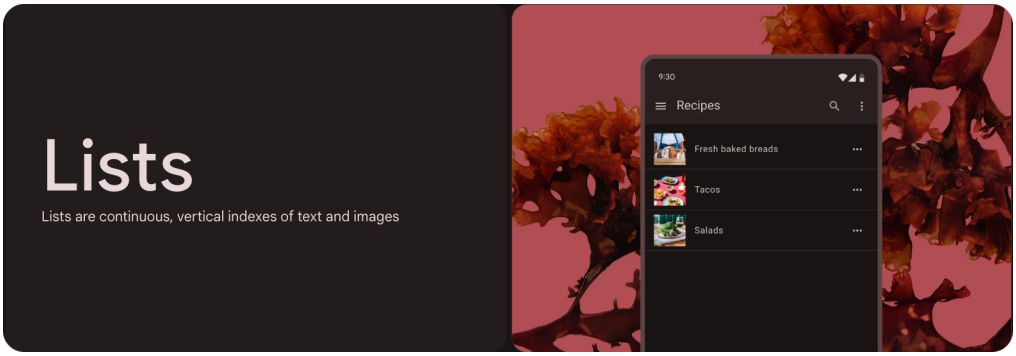
New in version 1.1.0.

2.3.22 List

See also:

Material Design spec, Lists

Lists are continuous, vertical indexes of text or images.

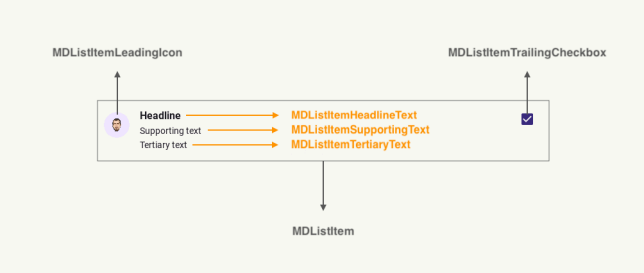


- Use lists to help users find a specific item and act on it;
- Order list items in logical ways (like alphabetical or numerical);
- Three sizes: one-line, two-line, and three-line;
- Keep items short and easy to scan;
- Show icons, text, and actions in a consistent format;

Usage

```
MDListItem:

    MDListItemLeadingIcon:  # MDListItemLeadingAvatar

    MDListItemHeadlineText:

    MDListItemSupportingText:

    MDListItemTertiaryText:

    MDListItemTrailingIcon:  # MDListItemTrailingCheckbox
```

Anatomy



Example:

One line list item

Declarative KV styles

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDListItem:
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint_x: .8

        MDListItemHeadlineText:
            text: "Headline"
'''


class Example(MDApp):
```

(continues on next page)

(continued from previous page)

```
    def build(self):
        return Builder.load_string(KV)


Example().run()
```

Declarative Python styles

```
from kivymd.uix.list import MDListItem, MDListItemHeadlineText
from kivymd.uix.screen import MDScreen
from kivymd.app import MDApp


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                MDListItem(
                    MDListItemHeadlineText(
                        text="Headline",
                    ),
                    pos_hint={"center_x": .5, "center_y": .5},
                    size_hint_x=0.8,
                ),
                md_bg_color=self.theme_cls.backgroundColor,
            )
        )


Example().run()
```

Two line list item

Declarative KV styles

```
MDListItem:

    MDListItemHeadlineText:
        text: "Headline"

    MDListItemSupportingText:
        text: "Supporting text"
```

Declarative Python styles

```
MDListItem(
    MDListItemHeadlineText(
        text="Headline",
    ),
    MDListItemSupportingText(
```

(continues on next page)

(continued from previous page)

```
        text="Supporting text",
    ),
)
```

Three line list item

Declarative KV styles

```
MDListItem:

    MDListItemHeadlineText:
        text: "Headline"

    MDListItemSupportingText:
        text: "Supporting text"

    MDListItemTertiaryText:
        text: "Tertiary text"
```

Declarative Python styles

```
MDListItem(
    MDListItemHeadlineText(
        text="Headline",
    ),
    MDListItemSupportingText(
        text="Supporting text",
    ),
    MDListItemTertiaryText(
        text="Tertiary text",
    ),
)
```

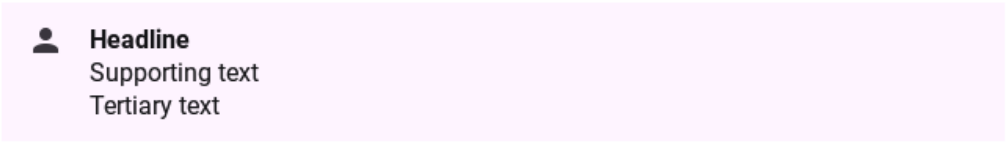
List item with leading icon

Declarative KV styles

```
MDListItem:

    MDListItemLeadingIcon:
        icon: "account"

    MDListItemHeadlineText:
        text: "Headline"

    MDListItemSupportingText:
        text: "Supporting text"

    MDListItemTertiaryText:
        text: "Tertiary text"
```

Declarative Python styles

```
MDListItem(
    MDListItemLeadingIcon(
        icon="account",
    ),
    MDListItemHeadlineText(
        text="Headline",
    ),
    MDListItemSupportingText(
        text="Supporting text",
    ),
    MDListItemTertiaryText(
        text="Tertiary text",
    ),
)
```



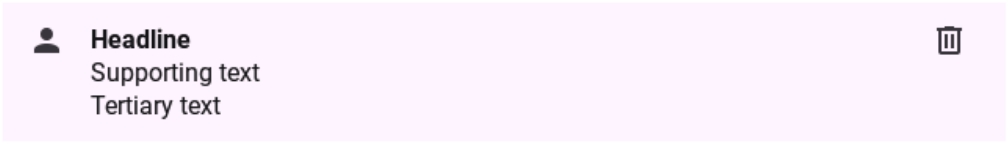
List item with trailing icon

Declarative KV styles

```
MDListItem:

    MDListItemLeadingIcon:
        icon: "account"

    MDListItemHeadlineText:
        text: "Headline"

    MDListItemSupportingText:
        text: "Supporting text"

    MDListItemTertiaryText:
        text: "Tertiary text"

    MDListItemTrailingIcon:
        icon: "trash-can-outline"
```

Declarative Python styles

```
MDListItem(
    MDListItemLeadingIcon(
        icon="account",
    ),
    MDListItemHeadlineText(
        text="Headline",
    ),
    MDListItemSupportingText(
        text="Supporting text",
    ),
    MDListItemTertiaryText(
        text="Tertiary text",
    ),
    MDListItemTrailingIcon(
        icon="trash-can-outline",
    ),
)
```



List item with trailing check

Declarative KV styles

```
MDListItem:

    MDListItemLeadingIcon:
        icon: "account"

    MDListItemHeadlineText:
        text: "Headline"

    MDListItemSupportingText:
        text: "Supporting text"

    MDListItemTertiaryText:
        text: "Tertiary text"

    MDListItemTrailingCheckbox:
```

Declarative Python styles

```
MDListItem(
    MDListItemLeadingIcon(
        icon="account",
    ),
    MDListItemHeadlineText(
        text="Headline",
    ),
    MDListItemSupportingText(
        text="Supporting text",
    ),
    MDListItemTertiaryText(
        text="Tertiary text",
    ),
    MDListItemTrailingCheckbox(
    ),
)
```

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.

For more information, see in the *MDGridLayout* class documentation.

class kivymd.uix.list.list.**BaseListItem**(**args*, ***kwargs*)

Base class for list items.

For more information, see in the *DeclarativeBehavior* and *BackgroundColorBehavior* and *RectangularRippleBehavior* and `ButtonBehavior` and *ThemableBehavior* and `StateLayerBehavior` classes documentation.

divider

Should I use divider for a list item.

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

divider_color

The divider color in (r, g, b, a) or string format.

divider_color is a `ColorProperty` and defaults to *None*.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the list item when the list item is disabled.

md_bg_color_disabled is a `ColorProperty` and defaults to *None*.

class kivymd.uix.list.list.**BaseListItemText**(**args*, ***kwargs*)

Base class for text labels of a list item.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.list.list.**BaseListItemIcon**(**args*, ***kwargs*)

Base class for leading/trailing icon of list item.

For more information, see in the *MDIcon* class documentation.

icon_color

Icon color in (r, g, b, a) or string format.

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

icon_color_disabled

The icon color in (r, g, b, a) or string format of the list item when the list item is disabled.

icon_color_disabled is a ColorProperty and defaults to *None*.

class kivymd.uix.list.list.**MDListItemHeadlineText**(**args*, ***kwargs*)

Implements a class for headline text of list item.

For more information, see in the *BaseListItemText* class documentation.

class kivymd.uix.list.list.**MDListItemSupportingText**(**args*, ***kwargs*)

Implements a class for secondary text of list item.

For more information, see in the *BaseListItemText* class documentation.

class kivymd.uix.list.list.**MDListItemTertiaryText**(**args*, ***kwargs*)

Implements a class for tertiary text of list item.

For more information, see in the *BaseListItemText* class documentation.

class kivymd.uix.list.list.**MDListItemTrailingSupportingText**(**args*, ***kwargs*)

Implements a class for trailing text of list item.

For more information, see in the *BaseListItemText* class documentation.

class kivymd.uix.list.list.**MDListItemLeadingIcon**(**args*, ***kwargs*)

Implements a class for leading icon class.

For more information, see in the *BaseListItemIcon* class documentation.

class kivymd.uix.list.list.**MDListItemLeadingAvatar**(***kwargs*)

Implements a class for leading avatar class.

For more information, see in the *ThemableBehavior* and *CircularRippleBehavior* and ButtonBehavior and *FitImage* classes documentation.

class kivymd.uix.list.list.**MDListItemTrailingIcon**(**args*, ***kwargs*)

Implements a class for trailing icon class.

For more information, see in the *BaseListItemIcon* class documentation.

class kivymd.uix.list.list.**MDListItemTrailingCheckbox**(***kwargs*)

Implements a class for trailing checkbox class.

For more information, see in the *MDCheckbox* class documentation.

class kivymd.uix.list.list.**MDListItem**(**args*, ***kwargs*)

Implements a list item.

For more information, see in the *BaseListItem* and BoxLayout classes documentation.

add_widget(*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.

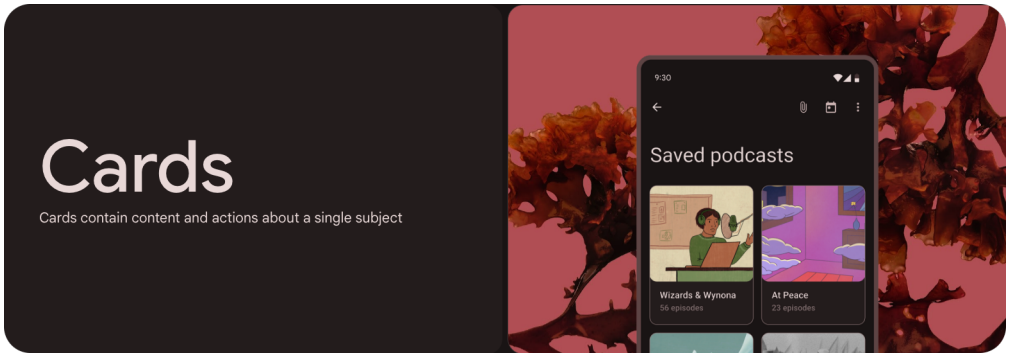
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.23 Card

See also:

Material Design 3 spec, Cards

Cards contain content and actions about a single subject.



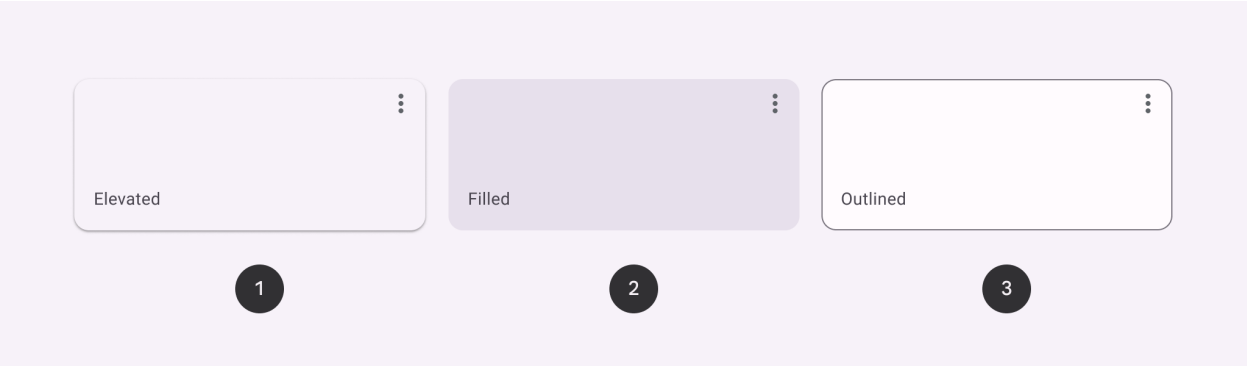
KivyMD provides the following card classes for use:

- *MDCard*
- *MDCardSwipe*

Note: *MDCard* inherited from BoxLayout. You can use all parameters and attributes of the BoxLayout class in the *MDCard* class.

MDCard

There are three types of cards: elevated, filled, and outlined:



1. Elevated card
2. Filled card
3. Outlined card

Example

Declarative KV and imperative python styles

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

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

KV = '''
<MyCard>
    padding: "4dp"
    size_hint: None, None
    size: "240dp", "100dp"

    MDRelativeLayout:

        MDIconButton:
            icon: "dots-vertical"
            pos_hint: {"top": 1, "right": 1}

        MDLabel:
            text: root.text
            adaptive_size: True
            color: "grey"
            pos: "12dp", "12dp"
            bold: True


MDScreen:
```

(continues on next page)

(continued from previous page)

```
    theme_bg_color: "Custom"
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        id: box
        adaptive_size: True
        spacing: "12dp"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class MyCard(MDCard):
    '''Implements a material card.'''

    text = StringProperty()


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

    def on_start(self):
        for style in ("elevated", "filled", "outlined"):
            self.root.ids.box.add_widget(
                MyCard(style=style, text=style.capitalize())
            )


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.card import MDCard
from kivymd.uix.label import MDLabel
from kivymd.uix.relativelayout import MDRelativeLayout
from kivymd.uix.screen import MDScreen


class MyCard(MDCard):
    '''Implements a material card.'''


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                MDBoxLayout(
                    id="box",
                    adaptive_size=True,
```

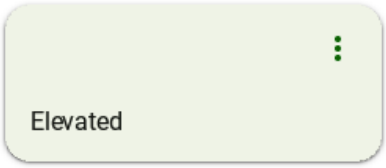
(continues on next page)

(continued from previous page)

```
                    spacing="12dp",
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                ),
                theme_bg_color="Custom",
                md_bg_color=self.theme_cls.backgroundColor,
            )
        )

    def on_start(self):
        for style in ("elevated", "filled", "outlined"):
            self.root.ids.box.add_widget(
                MyCard(
                    MDRelativeLayout(
                        MDIconButton(
                            icon="dots-vertical",
                            pos_hint={"top": 1, "right": 1}
                        ),
                        MDLabel(
                            text=style.capitalize(),
                            adaptive_size=True,
                            pos=("12dp", "12dp"),
                        ),
                    ),
                    style=style,
                    padding="4dp",
                    size_hint=(None, None),
                    size=("240dp", "100dp"),
                    ripple_behavior=True,
                )
            )


Example().run()
```

Elevated

```
MDCard
    style: "elevated"
```



Filled

```
MDCard
    style: "filled"
```

Outlined

```
MDCard
    style: "outlined"
```

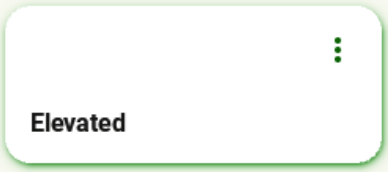
Customization of card

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    theme_bg_color: "Custom"
    md_bg_color: self.theme_cls.backgroundColor

    MDCard:
        style: "elevated"
        pos_hint: {"center_x": .5, "center_y": .5}
        padding: "4dp"
        size_hint: None, None
        size: "240dp", "100dp"
        # Sets custom properties.
        theme_shadow_color: "Custom"
        shadow_color: "green"
        theme_bg_color: "Custom"
        md_bg_color: "white"
        md_bg_color_disabled: "grey"
        theme_shadow_offset: "Custom"
        shadow_offset: (1, -2)
        theme_shadow_softness: "Custom"
        shadow_softness: 1
        theme_elevation_level: "Custom"
        elevation_level: 2
```

(continues on next page)

(continued from previous page)

```
        MDRelativeLayout:

            MDIconButton:
                icon: "dots-vertical"
                pos_hint: {"top": 1, "right": 1}

            MDLabel:
                text: "Elevated"
                adaptive_size: True
                color: "grey"
                pos: "12dp", "12dp"
                bold: True
'''


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


Example().run()
```



MDCardSwipe

To create a card with *swipe-to-delete* behavior, you must create a new class that inherits from the *MDCardSwipe* class:

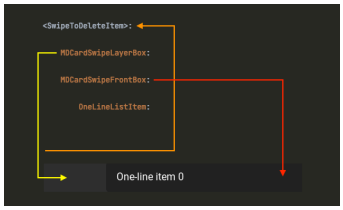
```
<SwipeToDeleteItem>
    size_hint_y: None
    height: content.height

    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()
```



Example

Declarative KV and imperative python styles

```
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:

    MDScrollView:
```

(continues on next page)

(continued from previous page)

```
        MDList:
            id: md_list
            padding: 0
'''


class SwipeToDeleteItem(MDCardSwipe):
    text = StringProperty()


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        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}")
            )


Example().run()
```

Declarative python styles

```
from kivymd.app import MDApp
from kivymd.uix.card import (
    MDCardSwipe, MDCardSwipeLayerBox, MDCardSwipeFrontBox
)
from kivymd.uix.list import MDList, OneLineListItem
from kivymd.uix.screen import MDScreen
from kivymd.uix.scrollview import MDScrollView


class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDScreen(
                MDScrollView(
                    MDList(
```

(continues on next page)

(continued from previous page)

```
                        id="md_list",
                        padding=0,
                    ),
                    id="scroll",
                    scroll_timeout=100,
                ),
            )
        )

    def on_start(self):
        '''Creates a list of cards.'''

        for i in range(20):
            self.root.ids.scroll.ids.md_list.add_widget(
                MDCardSwipe(
                    MDCardSwipeLayerBox(),
                    MDCardSwipeFrontBox(
                        OneLineListItem(
                            id="content",
                            text=f"One-line item {i}",
                            _no_ripple_effect=True,
                        )
                    ),
                    size_hint_y=None,
                    height="48dp",
                )
            )


Example().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"  # "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:

Declarative KV styles

```
<SwipeToDeleteItem>:
    on_swipe_complete: app.on_swipe_complete(root)
```

Declarative python styles

```
.. code-block:: python

    MDCardSwipe(
        ...
        on_swipe_complete=self.on_swipe_complete,
    )
```

Imperative python styles

```
def on_swipe_complete(self, instance):
    self.root.ids.md_list.remove_widget(instance)
```

Decralative python styles

```
def on_swipe_complete(self, instance):
    self.root.ids.box.ids.scroll.ids.md_list.remove_widget(instance)
```

Add content to the bottom layer of the card

To add content to the bottom layer of the card, use the *MDCardSwipeLayerBox* class.

```
<SwipeToDeleteItem>:

    MDCardSwipeLayerBox:
        padding: "8dp"

        MDIconButton:
            icon: "trash-can"
            pos_hint: {"center_y": .5}
            on_release: app.remove_item(root)
```

API - kivymd.uix.card.card

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

Card class.

For more information, see in the *DeclarativeBehavior* and *MDAdaptiveWidget* and *ThemableBehavior* and *BackgroundColorBehavior* and *CommonElevationBehavior* and *RectangularRippleBehavior* and *StateLayerBehavior* and ButtonBehavior and BoxLayout and classes documentation.

ripple_behavior

Use ripple effect for card.

ripple_behavior is a BooleanProperty and defaults to *False*.

radius

Card radius by default.

New in version 1.0.0.

radius is an VariableListProperty and defaults to *[dp(16), dp(16), dp(16), dp(16)]*.

style

Card type.

New in version 1.0.0.

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

style is an OptionProperty and defaults to *None*.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the card when the card is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

on_press(**args*) → None

Fired when the button is pressed.

on_release(**args*) → None

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

on_ripple_behavior(*interval: int | float*, *value: bool*) → None

Fired when the *ripple_behavior* value changes.

set_properties_widget() → None

Fired *on_release/on_press/on_enter/on_leave* events.

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

Card swipe class.

For more information, see in the *MDRelativeLayout* class documentation.

Events

on_swipe_complete
Fired 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'*.

closing_interval

Interval for closing the front layer.

New in version 1.1.0.

closing_interval is a NumericProperty and defaults to *0*.

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*.

on_swipe_complete(**args*)

Fired when a swipe of card is completed.

on_anchor(*instance_swipe_to_delete_item*, *anchor_value: str*) → None

Fired when the value of *anchor* changes.

on_open_progress(*instance_swipe_to_delete_item*, *progress_value: float*) → None

Fired when the value of *open_progress* changes.

on_touch_move(*touch*)

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

See *on_touch_down()* for more information.

on_touch_up(*touch*)

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

See *on_touch_down()* for more information.

on_touch_down(*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.

open_card() → None

Animates the opening of the card.

close_card(**args*) → None

Animates the closing of the card.

add_widget(*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.card.card.**MDCardSwipeFrontBox**(**args*, ***kwargs*)

Card swipe front box.

For more information, see in the *MDCard* class documentation.

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

Card swipe back box.

For more information, see in the *MDBoxLayout* class documentation.

2.3.24 SegmentedButton

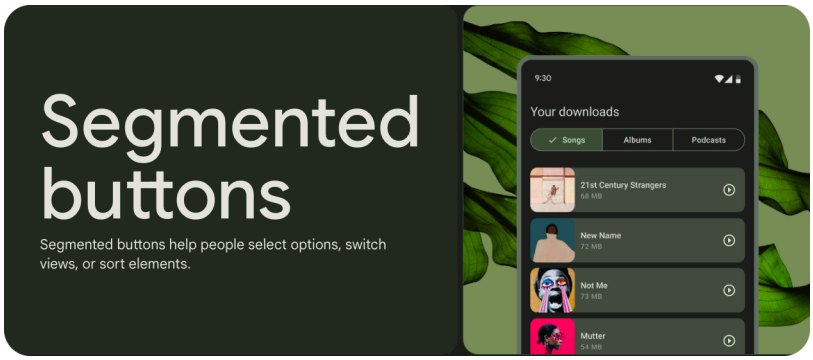
New in version 1.2.0.

See also:

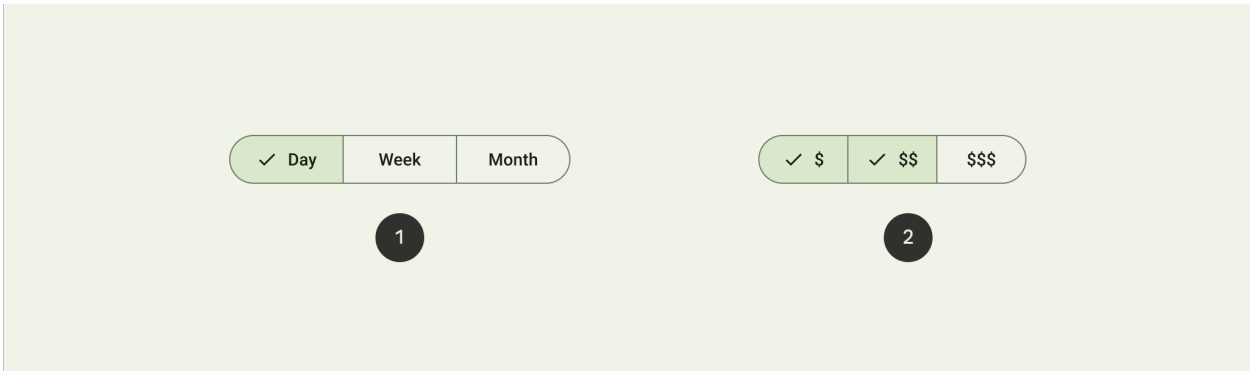
Material Design spec, Segmented buttons

Segmented control

Segmented buttons help people select options, switch views, or sort elements.

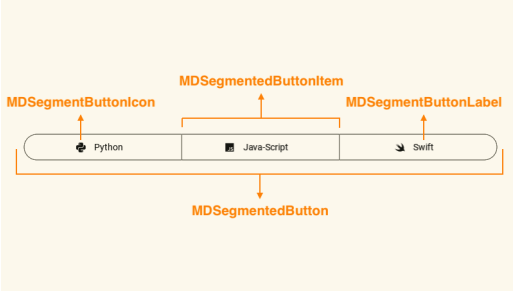


- Segmented buttons can contain icons, label text, or both
- Two types: single-select and multi-select
- Use for simple choices between two to five items (for more items or complex choices, use chips)



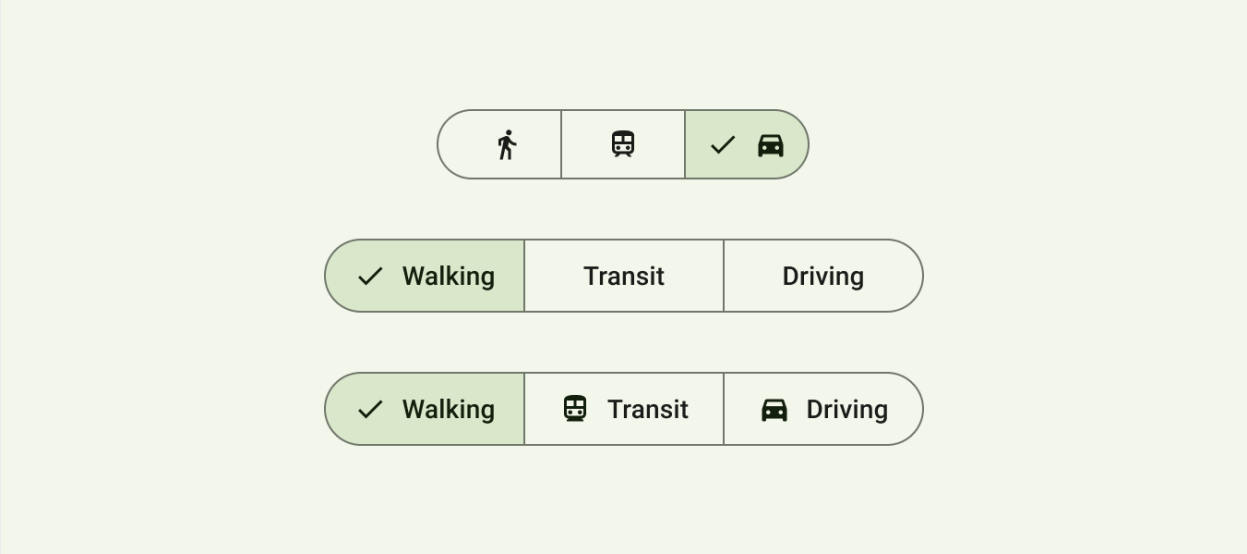
1. Single-select segmented button
2. Multi-select segmented button

Anatomy



Icons

Icons may be used as labels by themselves or alongside text. If an icon is used without label text, it must clearly communicate the option it represents.



Use with text and icon

```
MDSegmentedButton:

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-python"

        MDSegmentButtonLabel:
            text: "Python"

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-javascript"

        MDSegmentButtonLabel:
            text: "Java-Script"

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-swift"

        MDSegmentButtonLabel:
            text: "Swift"
```

Use without text with an icon

```
MDSegmentedButton:

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-python"

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-javascript"

    MDSegmentedButtonItem:

        MDSegmentButtonIcon:
            icon: "language-swift"
```

Use only text

```
MDSegmentedButton:

    MDSegmentedButtonItem:

        MDSegmentButtonLabel:
            text: "Python"

    MDSegmentedButtonItem:

        MDSegmentButtonLabel:
            text: "Java-Script"

    MDSegmentedButtonItem:

        MDSegmentButtonLabel:
            text: "Swift"
```

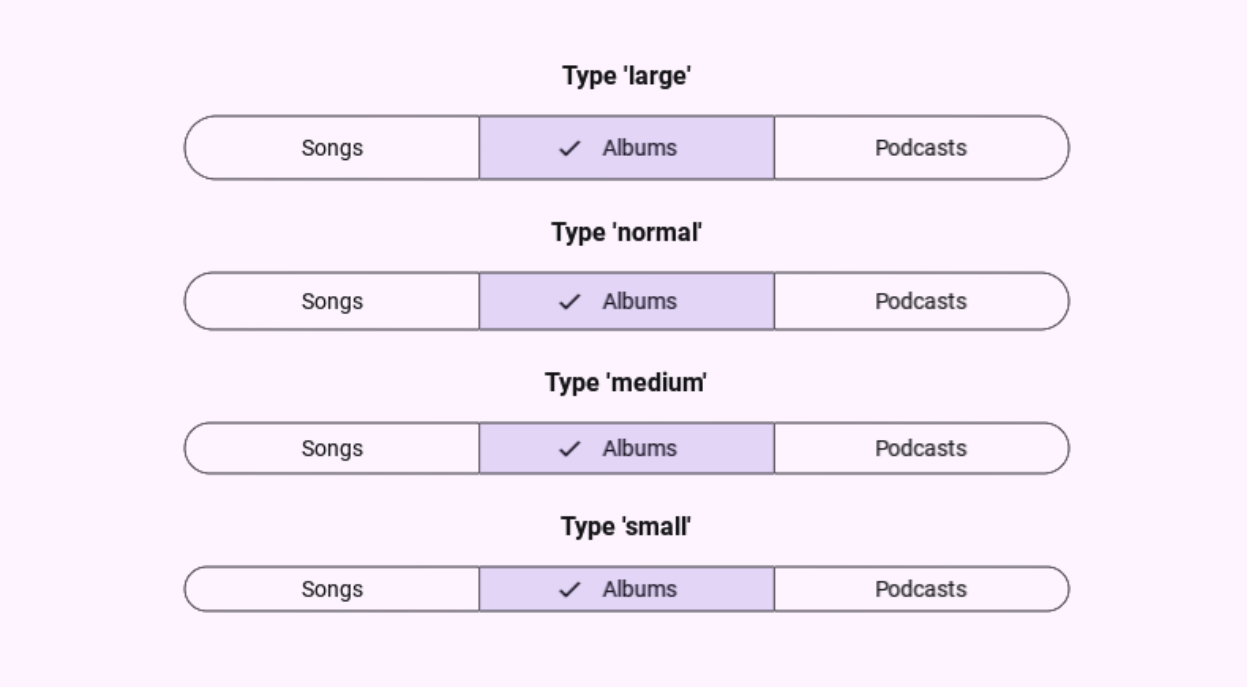
Multiselect

For multiple marking of elements, use the *kivymd.uix.segmentedbutton.segmentedbutton.MDSegmentedButton.multiselect* parameter:

```
MDSegmentedButton:
    multiselect: True
```

Type

Density can be used in denser UIs where space is limited. Density is only applied to the height. Each step down in density removes 4dp from the height.




```
from kivy.lang import Builder

from kivymd.uix.label import MDLabel
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.segmentedbutton import (
    MDSegmentedButton,
    MDSegmentedButtonItem,
    MDSegmentButtonLabel,
)
from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        id: box
        orientation: "vertical"
        size_hint_x: .7
        adaptive_height: True
        spacing: "24dp"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class Example(MDApp):
    def on_start(self):
        for segment_type in ["large", "normal", "medium", "small"]:
            self.root.ids.box.add_widget(
                MDBoxLayout(
                    MDLabel(
```

(continues on next page)

(continued from previous page)

```
                        text=f"Type '{segment_type}'",
                        adaptive_height=True,
                        bold=True,
                        pos_hint={"center_y": 0.5},
                        halign="center",
                    ),
                    MDSegmentedButton(
                        MDSegmentedButtonItem(
                            MDSegmentButtonLabel(
                                text="Songs",
                            ),
                        ),
                        MDSegmentedButtonItem(
                            MDSegmentButtonLabel(
                                text="Albums",
                            ),
                        ),
                        MDSegmentedButtonItem(
                            MDSegmentButtonLabel(
                                text="Podcasts",
                            ),
                        ),
                        type=segment_type,
                    ),
                    orientation="vertical",
                    spacing="12dp",
                    adaptive_height=True,
                )
            )

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

A practical example

```
import os

from faker import Faker

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

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout

import asynckivy
```

(continues on next page)

(continued from previous page)

```
KV = '''
<UserCard>
    adaptive_height: True
    radius: 16

    MDListItem:
        radius: 16
        theme_bg_color: "Custom"
        md_bg_color: self.theme_cls.secondaryContainerColor

        MDListItemLeadingAvatar:
            source: root.album

        MDListItemHeadlineText:
            text: root.name

        MDListItemSupportingText:
            text: root.path_to_file


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        orientation: "vertical"
        padding: "12dp"
        spacing: "12dp"

        MDLabel:
            adaptive_height: True
            text: "Your downloads"
            theme_font_style: "Custom"
            font_style: "Display"
            role: "small"

        MDSegmentedButton:
            size_hint_x: 1

            MDSegmentedButtonItem:
                on_active: app.generate_card()

                MDSegmentButtonLabel:
                    text: "Songs"
                    active: True

            MDSegmentedButtonItem:
                on_active: app.generate_card()

                MDSegmentButtonLabel:
                    text: "Albums"
```

(continues on next page)

(continued from previous page)

```
            MDSegmentedButtonItem:
                on_active: app.generate_card()

                MDSegmentButtonLabel:
                    text: "Podcasts"

        RecycleView:
            id: card_list
            viewclass: "UserCard"
            bar_width: 0

            RecycleBoxLayout:
                orientation: 'vertical'
                spacing: "16dp"
                padding: "16dp"
                default_size: None, dp(72)
                default_size_hint: 1, None
                size_hint_y: None
                height: self.minimum_height
'''


class UserCard(MDBoxLayout):
    name = StringProperty()
    path_to_file = StringProperty()
    album = StringProperty()


class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)

    def generate_card(self):
        async def generate_card():
            for i in range(10):
                await asynckivy.sleep(0)
                self.root.ids.card_list.data.append(
                    {
                        "name": fake.name(),
                        "path_to_file": f"{os.path.splitext(fake.file_path())[0]}.mp3",
                        "album": fake.image_url(),
                    }
                )

        fake = Faker()
        self.root.ids.card_list.data = []
        Clock.schedule_once(lambda x: asynckivy.start(generate_card()))


Example().run()
```

API break

1.2.0 version

```
MDSegmentedButton:
    on_marked: func(*args)

    MDSegmentedButtonItem:
        icon: ...
        text: ...
```

2.0.0 version

```
MDSegmentedButton:

    MDSegmentedButtonItem:
        on_active: func(*args)

        MDSegmentButtonIcon:
            icon: ...

        MDSegmentButtonLabel:
            text: ...
```

API - kivymd.uix.segmentedbutton.segmentedbutton

class kivymd.uix.segmentedbutton.segmentedbutton.**MDSegmentedButtonItem**(**args*, ***kwargs*)

Segment button item.

For more information see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and *RectangularRippleBehavior* and ButtonBehavior and *StateLayerBehavior* and RelativeLayout and class documentation.

selected_color

Color of the marked segment.

New in version 2.0.0.

selected_color is a ColorProperty and defaults to *None*.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the list item when the list item is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

active

Background color of an disabled segment.

active is an BooleanProperty and defaults to *False*.

add_widget(*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_line_color(*instance*, *value*) → None

Fired when the values of `line_color` change.

on_active(*instance*, *value*) → None

Fired when the *active* value changes. Animates the marker icon for the element.

on_disabled(*instance*, *value*) → None

Fired when the `disabled` value changes.

class kivymd.uix.segmentedbutton.segmentedbutton.**MDSegmentedButton**(**args*, ***kwargs*)

Segment button panel.

For more information, see in the *MDBoxLayout* class documentation.

multiselect

Do I allow multiple segment selection.

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

hiding_icon_transition

Name of the transition hiding the current icon.

hiding_icon_transition is a `StringProperty` and defaults to *'linear'*.

hiding_icon_duration

Duration of hiding the current icon.

hiding_icon_duration is a `NumericProperty` and defaults to *1*.

opening_icon_transition

The name of the transition that opens a new icon of the "marked" type.

opening_icon_transition is a StringProperty and defaults to *'linear'*.

opening_icon_duration

The duration of opening a new icon of the "marked" type.

opening_icon_duration is a NumericProperty and defaults to *0.1*.

selected_segments

The list of *MDSegmentedButtonItem* objects that are currently marked.

selected_segments is a ListProperty and defaults to *[]*.

type

Density can be used in denser UIs where space is limited. Density is only applied to the height. Each step down in density removes '4dp' from the height.

New in version 2.0.0.

Available options are: 'large', 'normal', 'medium', 'small'.

type is an OptionProperty and defaults to *'large'*.

selected_icon_color

Color in (r, g, b, a) or string format of the icon of the marked segment.

New in version 2.0.0.

selected_icon_color is a ColorProperty and defaults to *None*.

get_marked_items() → list

Returns a list of active item objects.

get_items() → list

Returns a list of item objects.

adjust_segment_radius(**args*) → None

Rounds off the first and last elements.

mark_item(*segment_item:* MDSegmentedButtonItem) → None

Fired when a segment element is clicked (*on_release* event).

add_widget(*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)
```

class kivymd.uix.segmentedbutton.segmentedbutton.**MDSegmentButtonIcon**(**args*, ***kwargs*)

Implements a icon for *MDSegmentedButtonItem* class.

New in version 2.0.0.

For more information, see in the *MDIcon* class documentation.

class kivymd.uix.segmentedbutton.segmentedbutton.**MDSegmentButtonLabel**(**args*, ***kwargs*)

Implements a label for *MDSegmentedButtonItem* class.

New in version 2.0.0.

For more information, see in the *MDLabel* class documentation.

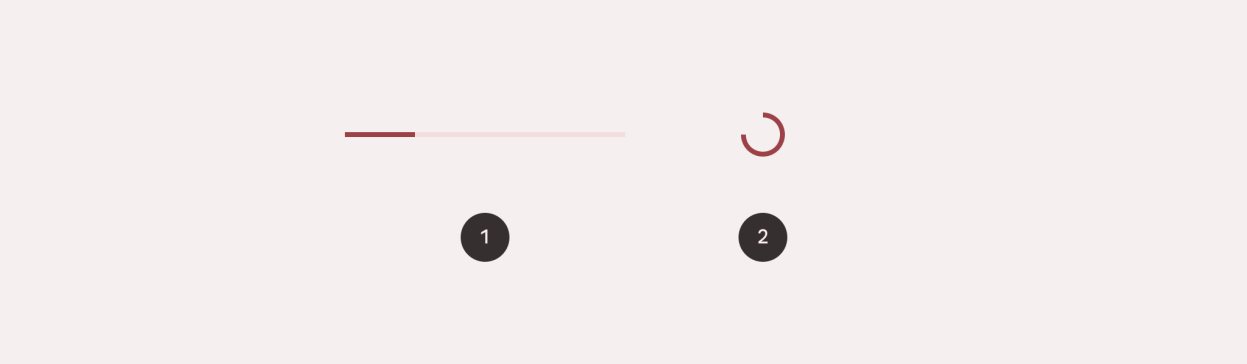
2.3.25 ProgressIndicator

See also:

Material Design spec, ProgressIndicator

Progress indicators show the status of a process in real time.

- Use the same progress indicator for all instances of a process (like loading)
- Two types: linear and circular
- Never use them as decoration
- They capture attention through motion

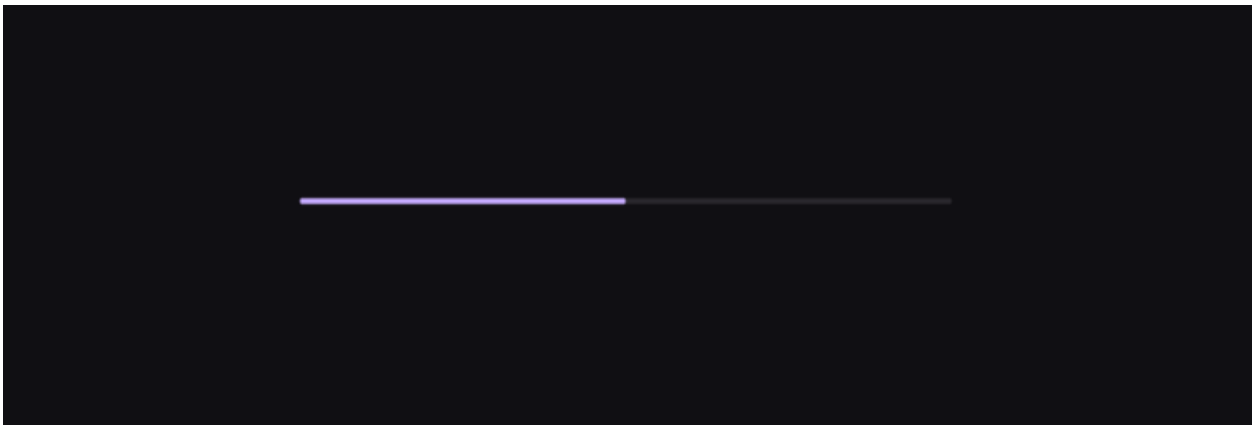


1. Linear progress indicator
2. Circular progress indicator

Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDLinearProgressIndicator:
        size_hint_x: .5
        value: 50
        pos_hint: {'center_x': .5, 'center_y': .4}
'''


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

Example().run()



```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDCircularProgressIndicator:
        size_hint: None, None
        size: "48dp", "48dp"
```

(continues on next page)

(continued from previous page)

```
        pos_hint: {'center_x': .5, 'center_y': .5}
'''


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


Example().run()
```

Linear progress indicators can be determinate or indeterminate.

Determinate linear progress indicator

Determinate operations display the indicator increasing from 0 to 100% of the track, in sync with the process's progress.

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDLinearProgressIndicator:
        id: progress
        size_hint_x: .5
        type: "determinate"
        pos_hint: {'center_x': .5, 'center_y': .4}
'''


class Example(MDApp):
    def on_start(self):
        self.root.ids.progress.start()

    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)


Example().run()
```

Circular progress indicators can be determinate or indeterminate.

Indeterminate circular progress indicator

Indeterminate operations display the indicator continually growing and shrinking along the track until the process is complete..

```
MDCircularProgressIndicator:
    size_hint: None, None
    size: "48dp", "48dp"
```

Determinate circular progress indicator

```
MDCircularProgressIndicator:
    size_hint: None, None
    size: "48dp", "48dp"
    determinate: True
    on_determinate_complete: print(args)
```

API break

1.1.1 version

```
MDProgressBar:
    value: 50
    color: app.theme_cls.accent_color
```

```
MDSpinner:
    size_hint: None, None
    size: dp(48), dp(48)
```

2.0.0 version

```
MDLinearProgressIndicator:
    value: 50
    indicator_color: app.theme_cls.errorColor
```

```
MDCircularProgressIndicator:
    size_hint: None, None
    size: dp(48), dp(48)
```

API - kivymd.uix.progressindicator.progressindicator

class kivymd.uix.progressindicator.progressindicator.**MDLinearProgressIndicator**(***kwargs*)

Implementation of the linear progress indicator.

For more information, see in the *ThemableBehavior* and ProgressBar classes documentation.

radius

Progress line radius.

New in version 1.2.0.

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

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'*.

indicator_color

Color of the active track.

Changed in version 2.0.0: Rename from *color* to *indicator_color* attribute.

indicator_color is an ColorProperty and defaults to *None*.

track_color

Progress bar back color in (r, g, b, a) or string format.

New in version 1.0.0.

Changed in version 2.0.0: Rename from *back_color* to *track_color* attribute.

track_color is an ColorProperty and defaults to *None*.

running_determinate_transition

Running transition.

Changed in version 2.0.0: Rename from *running_transition* to *running_determinate_transition* attribute.

running_determinate_transition is an StringProperty and defaults to *'out_quart'*.

catching_determinate_transition

Catching transition.

Changed in version 2.0.0: Rename from *catching_transition* to *catching_determinate_transition* attribute.

catching_determinate_transition is an StringProperty and defaults to *'out_quart'*.

running_determinate_duration

Running duration.

Changed in version 2.0.0: Rename from *running_duration* to *running_determinate_duration* attribute.

running_determinate_duration is an NumericProperty and defaults to *2.5*.

catching_determinate_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*.

running_indeterminate_transition

Running transition.

running_indeterminate_transition is an `StringProperty` and defaults to *'in_cubic'*.

catching_indeterminate_transition

Catching transition.

catching_indeterminate_transition is an `StringProperty` and defaults to *'out_quart'*.

running_indeterminate_duration

Running duration.

running_indeterminate_duration is an `NumericProperty` and defaults to *0.5*.

catching_indeterminate_duration

Catching duration.

catching_indeterminate_duration is an `NumericProperty` and defaults to *0.8*.

check_size(**args*) → None

start() → None

Start animation.

stop() → None

Stop animation.

running_away(**args*) → None

catching_up(**args*) → None

on_value(*instance*, *value*)

class kivymd.uix.progressindicator.progressindicator.**MDCircularProgressIndicator**(***kwargs*)

Implementation of the circular progress indicator.

Changed in version 2.0.0: Rename *MDSpinner* to *MDCircularProgressIndicator* class.

For more information, see in the *ThemableBehavior* and `Widget` classes documentation.

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 indicator 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 indicator.

line_width is a NumericProperty and defaults to *dp(2.25)*.

active

Use *active* to start or stop the indicator.

active is a BooleanProperty and defaults to *True*.

color

Indicator color in (r, g, b, a) or string format.

color is a ColorProperty and defaults to *None*.

palette

A set of colors. Changes with each completed indicator cycle.

palette is a ListProperty and defaults to *[]*.

on__rotation_angle(**args*)

on_palette(*instance*, *palette_list: list*) → None

Fired when the *palette* value changes.

on_active(*instance*, *value*) → None

Fired when the *active* value changes.

on_determinate_complete(**args*) → None

The event is fired at the end of the indicator loop in the *determinate = True* mode.

check_determinate(**args*) → None

Fired when the class is initialized.

2.3.26 FitImage

Example

Declarative KV styles

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
```

(continues on next page)

(continued from previous page)

```
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        radius: "36dp"
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint: .4, .8
        md_bg_color: self.theme_cls.onSurfaceVariantColor

        FitImage:
            source: "image.png"
            size_hint_y: .35
            pos_hint: {"top": 1}
            radius: "36dp", "36dp", 0, 0
'''


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


Example().run()
```

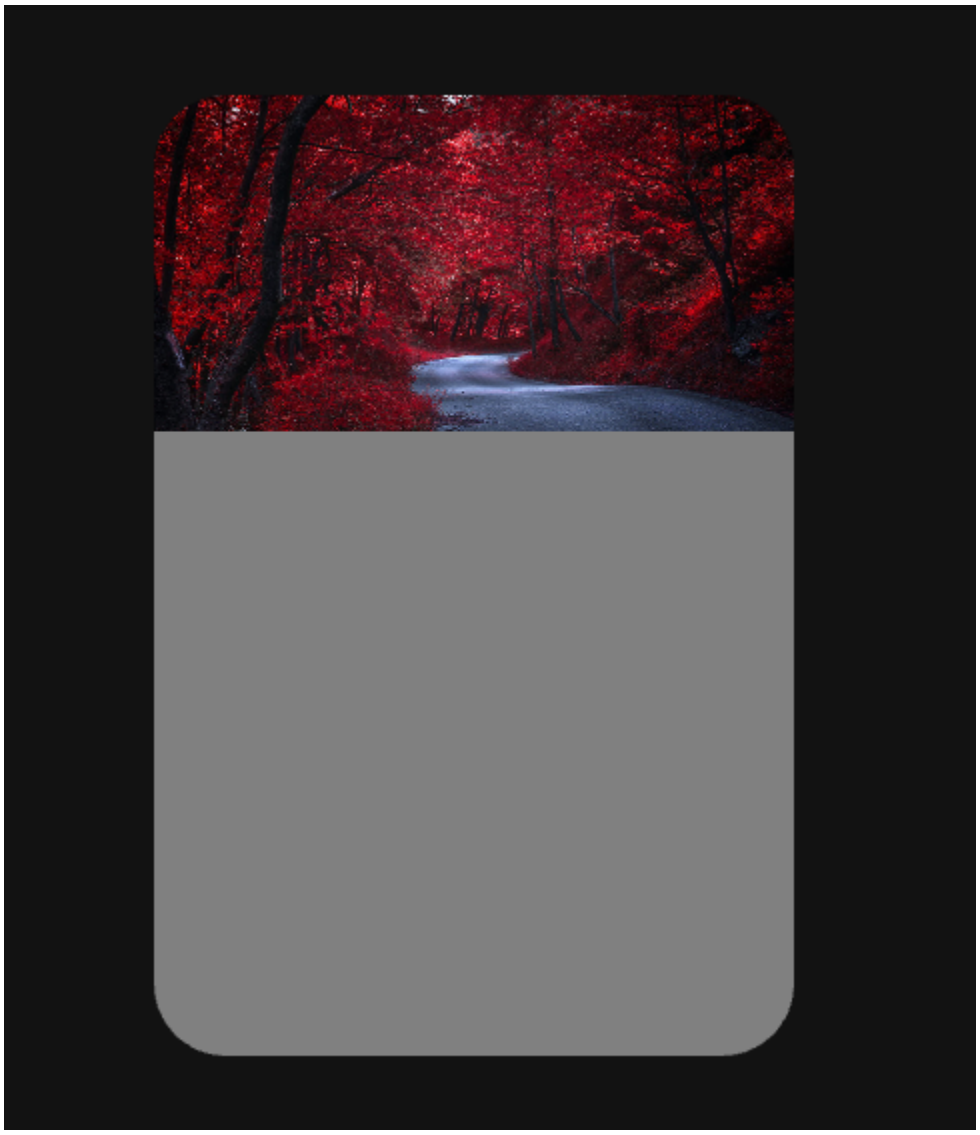
Declarative python styles

```
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.card import MDCard
from kivymd.uix.fitimage import FitImage
from kivymd.uix.screen import MDScreen


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                MDBoxLayout(
                    FitImage(
                        source="image.png",
                        size_hint_y=0.35,
                        pos_hint={"top": 1},
                        radius=(dp(36), dp(36), 0, 0),
                    ),
                    radius=dp(36),
                    md_bg_color=self.theme_cls.onSurfaceVariantColor,
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                    size_hint=(0.4, 0.8),
                ),
            )
        )
```

(continues on next page)

(continued from previous page)

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



API - kivymd.uix.fitimage.fitimage

class kivymd.uix.fitimage.fitimage.**FitImage**(***kwargs*)

Fit image class.

For more information, see in the AsyncImage and *StencilBehavior* classes documentation.

fit_mode

Image will be stretched horizontally or vertically to fill the widget box, **maintaining its aspect ratio**. If the image has a different aspect ratio than the widget, then the image will be clipped to fit.

fit_mode is a OptionProperty and defaults to *'cover'*.

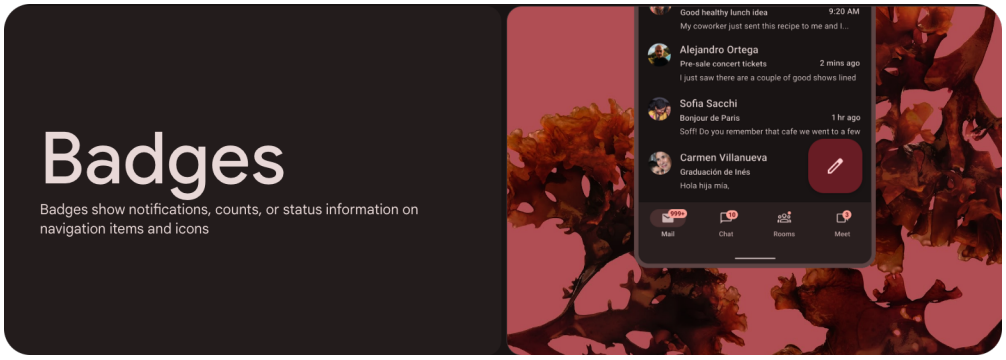
2.3.27 Badge

New in version 2.0.0.

See also:

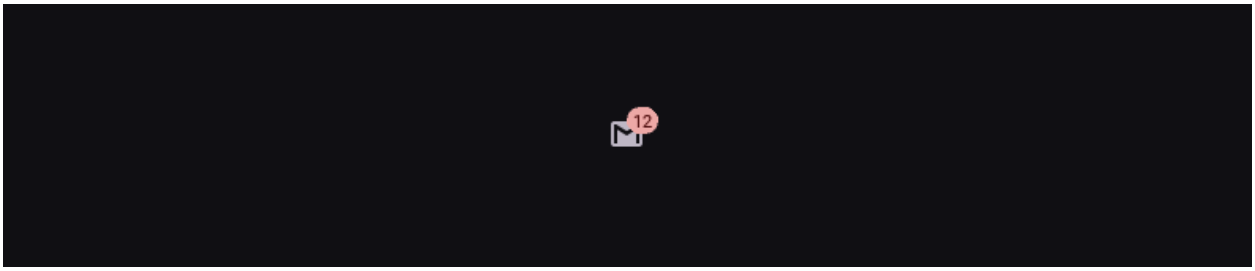
Material Design 3 spec, Badge

Badges show notifications, counts, or status information on navigation items and icons.



Example

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDIcon:
        icon: "gmail"
        pos_hint: {'center_x': .5, 'center_y': .5}

        MDBadge:
            text: "12"
'''


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


Example().run()
```



API - kivymd.uix.badge.badge

class kivymd.uix.badge.badge.**MDBadge**(**args*, ***kwargs*)

Badge class.

New in version 2.0.0.

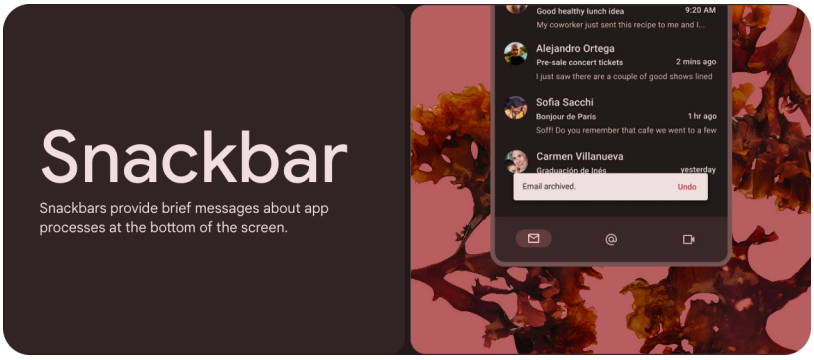
For more information see in the *MDLabel* class documentation.

2.3.28 Snackbar

See also:

Material Design spec, Snackbars

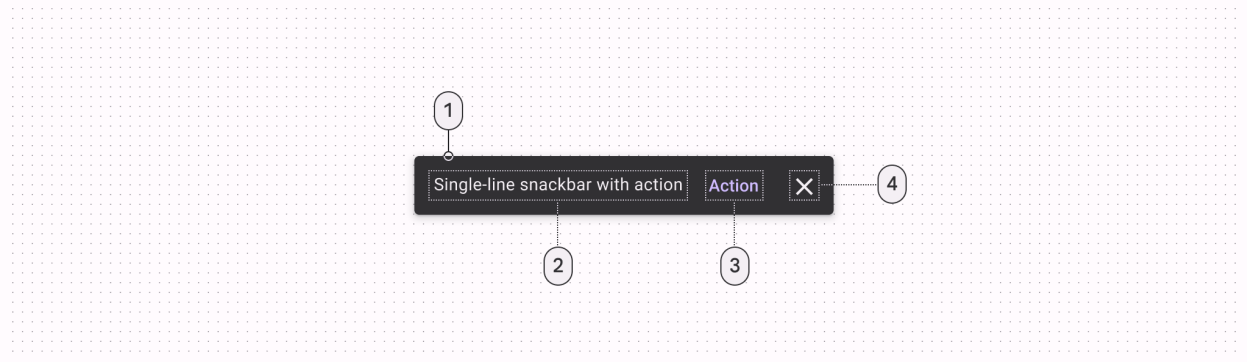
Snackbars provide brief messages about app processes at the bottom of the screen.



- Snackbars shouldn't interrupt the user's experience
- Usually appear at the bottom of the UI
- Can disappear on their own or remain on screen until the user takes action

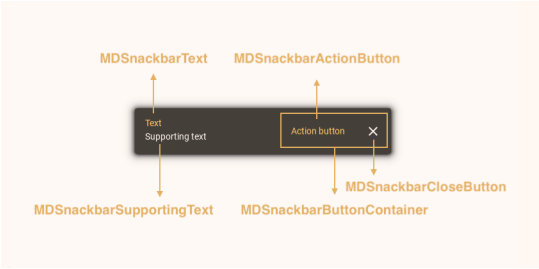
Usage

```
MDSnackbar(
    MDSnackbarText(
        text="Text",
    ),
    y=dp(24),
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
).open()
```



1. Container
2. Supporting text
3. Action (optional)
4. Icon (optional close affordance)

Anatomy

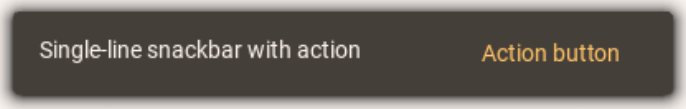


Configurations

1. Single line

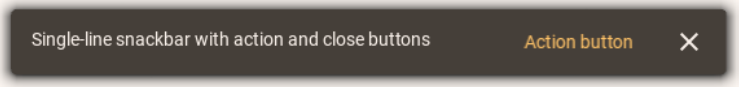

```
MDSnackbar(
    MDSnackbarText(
        text="Single-line snackbar",
    ),
    y=dp(24),
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
).open()
```

2. Single-line snackbar with action



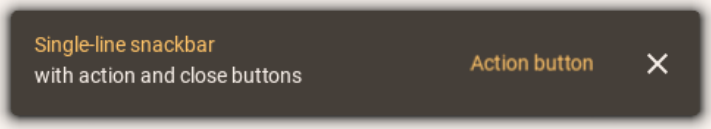

```
MDSnackbar(
    MDSnackbarSupportingText(
        text="Single-line snackbar with action",
    ),
    MDSnackbarButtonContainer(
        MDSnackbarActionButton(
            MDSnackbarActionButtonText(
                text="Action button"
            ),
        ),
        pos_hint={"center_y": 0.5}
    ),
    y=dp(24),
    orientation="horizontal",
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
).open()
```

3. Single-line snackbar with action and close buttons




```
MDSnackbar(
    MDSnackbarSupportingText(
        text="Single-line snackbar with action and close buttons",
    ),
    MDSnackbarButtonContainer(
        MDSnackbarActionButton(
            MDSnackbarActionButtonText(
                text="Action button"
            ),
        ),
        MDSnackbarCloseButton(
            icon="close",
        ),
        pos_hint={"center_y": 0.5}
    ),
    y=dp(24),
    orientation="horizontal",
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
).open()
```

4. Two-line snackbar with action and close buttons



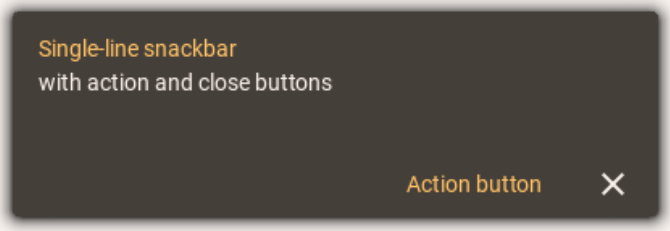

```
MDSnackbar(
    MDSnackbarText(
        text="Single-line snackbar",
    ),
    MDSnackbarSupportingText(
        text="with action and close buttons",
    ),
    MDSnackbarButtonContainer(
```

(continues on next page)

(continued from previous page)

```
        MDSnackbarActionButton(
            MDSnackbarActionButtonText(
                text="Action button"
            ),
        ),
        MDSnackbarCloseButton(
            icon="close",
        ),
        pos_hint={"center_y": 0.5}
    ),
    y=dp(24),
    orientation="horizontal",
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
).open()
```

5. Two-line snackbar with action and close buttons at the bottom




```
MDSnackbar(
    MDSnackbarText(
        text="Single-line snackbar with action",
    ),
    MDSnackbarSupportingText(
        text="and close buttons at the bottom",
        padding=[0, 0, 0, dp(56)],
    ),
    MDSnackbarButtonContainer(
        Widget(),
        MDSnackbarActionButton(
            MDSnackbarActionButtonText(
                text="Action button"
            ),
        ),
        MDSnackbarCloseButton(
            icon="close",
        ),
    ),
    y=dp(124),
```

(continues on next page)

(continued from previous page)

```
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
    padding=[0, 0, "8dp", "8dp"],
).open()
```

API break

1.1.1 version

```
snackbar = Snackbar(
    text="First string",
    snackbar_x="10dp",
    snackbar_y="24dp",
)
snackbar.size_hint_x = (
    Window.width - (snackbar.snackbar_x * 2)
) / Window.width
snackbar.buttons = [
    MDFlatButton(
        text="Done",
        theme_text_color="Custom",
        text_color="#8E353C",
        on_release=snackbar.dismiss,
    ),
]
snackbar.open()
```

1.2.0 version

```
MDSnackbar(
    MDLabel(
        text="First string",
    ),
    MDSnackbarActionButton(
        text="Done",
        theme_text_color="Custom",
        text_color="#8E353C",
    ),
    y=dp(24),
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
    md_bg_color="#E8D8D7",
).open()
```

2.0.0 version

```
MDSnackbar(
    MDSnackbarSupportingText(
        text="Single-line snackbar with action",
    ),
    MDSnackbarButtonContainer(
        MDSnackbarActionButton(
            MDSnackbarActionButtonText(
                text="Action button"
            ),
        ),
        pos_hint={"center_y": 0.5}
    ),
    y=dp(24),
    orientation="horizontal",
    pos_hint={"center_x": 0.5},
    size_hint_x=0.5,
    background_color=self.theme_cls.onPrimaryContainerColor,
).open()
```

API - kivymd.uix.snackbar.snackbar

class kivymd.uix.snackbar.snackbar.**MDSnackbarButtonContainer**(**args*, ***kwargs*)

The class implements a container for placing snackbar buttons.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

add_widget(*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)
```

class kivymd.uix.snackbar.snackbar.**MDSnackbarCloseButton**(***kwargs*)

Snackbar closed button class.

For more information, see in the *MDIconButton* class documentation.

class kivymd.uix.snackbar.snackbar.**MDSnackbarActionButtonText**(**args*, ***kwargs*)

The class implements the text for the *MDSnackbarActionButton* class.

Changed in version 2.2.0.

For more information, see in the *MDButtonText* class documentation.

class kivymd.uix.snackbar.snackbar.**MDSnackbarActionButton**(**args*, ***kwargs*)

Snackbar action button class.

For more information, see in the *MDButton* class documentation.

class kivymd.uix.snackbar.snackbar.**MDSnackbar**(**args*, ***kwargs*)

Snackbar class.

Changed in version 1.2.0: Rename *BaseSnackbar* to *MDSnackbar* class.

For more information, see in the *MotionShackBehavior* and *MDCard* and class documentation.

Events

on_open
Fired when a snackbar opened.

on_dismiss
Fired when a snackbar closes.

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*.

radius

Snackbar radius.

radius is a ListProperty and defaults to *[dp(4), dp(4), dp(4), dp(4)]*

background_color

The background color in (r, g, b, a) or string format of the snackbar.

background_color is a ColorProperty and defaults to *None*.

dismiss(**args*) → None

Dismiss the snackbar.

open() → None

Show the snackbar.

add_widget(*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_open(**args*) → None

Fired when a snackbar opened.

on_dismiss(**args*) → None

Fired when a snackbar closed.

class kivymd.uix.snackbar.snackbar.**MDSnackbarText**(**args*, ***kwargs*)

The class implements the text.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.snackbar.snackbar.**MDSnackbarSupportingText**(**args*, ***kwargs*)

The class implements the supporting text.

For more information, see in the *MDLabel* class documentation.

2.3.29 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 *MDScreenManager*.

For more information, see in the TransitionBase class documentation.

start(*instance_screen_manager:* kivymd.uix.screenmanager.MDScreenManager) → None

(internal) Starts the transition. This is automatically called by the ScreenManager.

animated_hero_in() → None

Animates the flight of heroes from screen **A** to screen **B**.

animated_hero_out() → None

Animates the flight of heroes from screen **B** to screen **A**.

on_complete() → None

Override method. See :attr:`kivy.uix.screenmanager.TransitionBase.on_complete'.

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(*instance_screen_manager:* kivymd.uix.screenmanager.MDScreenManager) → None

(internal) Starts the transition. This is automatically called by the ScreenManager.

on_progress(*progression: float*) → None

class kivymd.uix.transition.transition.**MDSharedAxisTransition**

Android default screen transition

transition_axis

Axis of the transition. Available values "x", "y", and "z".

transition_axis is an OptionProperty and defaults to *"x"*.

duration

Duration in seconds of the transition. Android recommends these intervals:

Table 1: Android transition values (in seconds)

Name	value
small_1	0.075
small_2	0.15
medium_1	0.2
medium_2	0.25
large_1	0.3
large_2	0.35

duration is a NumericProperty and defaults to 0.15 (= 150ms).

slide_distance

Distance to which it slides left, right, bottom or up depending on axis.

slide_distance is a NumericProperty and defaults to *dp(15)*.

opposite

Decides Transition direction.

opposite is a BooleanProperty and defaults to *False*.

start(*manager*)

(internal) Starts the transition. This is automatically called by the ScreenManager.

on_progress(*progress*)

on_complete()

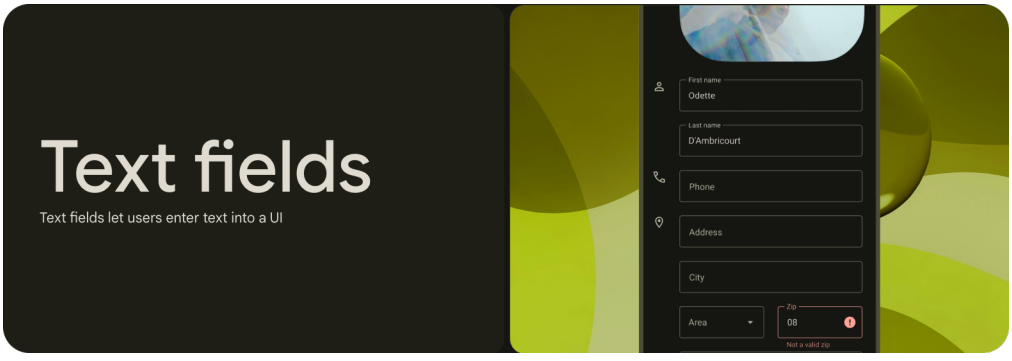
Override method. See :attr:`kivy.uix.screenmanager.TransitionBase.on_complete'.

2.3.30 Text fields

See also:

Material Design spec, Text fields

Text fields let users enter text into a UI.



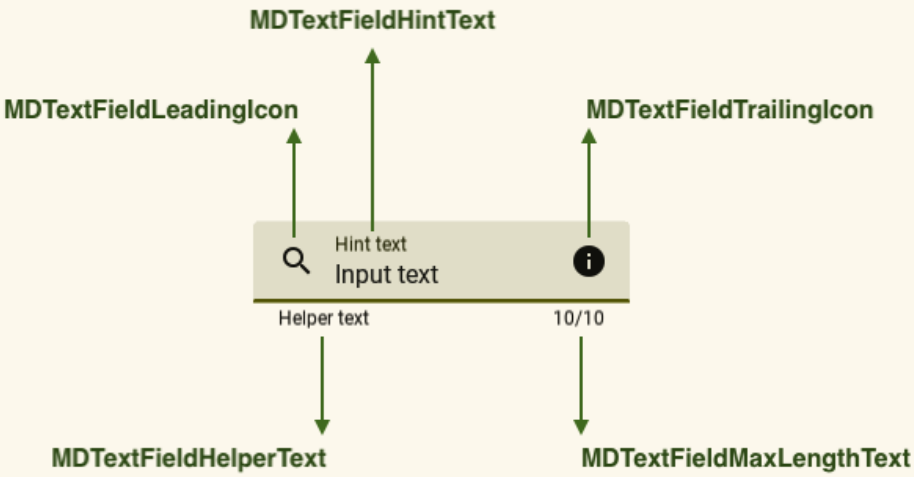
- Make sure text fields look interactive
- Two types: filled and outlined
- The text field's state (blank, with input, error, etc) should be visible at a glance
- Keep labels and error messages brief and easy to act on
- Text fields commonly appear in forms and dialogs

1. Filled text field
2. Outlined text field

Usage

```
MDTextField:
    mode: "filled"

    MDTextFieldLeadingIcon:
        icon: "magnify"

    MDTextFieldHintText:
        text: "Hint text"

    MDTextFieldHelperText:
        text: "Helper text"
        mode: "persistent"

    MDTextFieldTrailingIcon:
        icon: "information"

    MDTextFieldMaxLengthText:
        max_text_length: 10
```

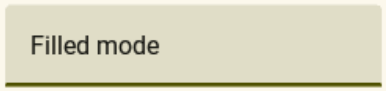
Anatomy



Available types of text fields

Filled mode

```
MDTextField:
    mode: "filled"
```



Outlined mode

```
MDTextField:
    mode: "outlined"
```

Example

Declarative KV style

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: app.theme_cls.backgroundColor

    MDTextField:
        mode: "outlined"
        size_hint_x: None
        width: "240dp"
        pos_hint: {"center_x": .5, "center_y": .5}

        MDTextFieldLeadingIcon:
            icon: "account"

        MDTextFieldHintText:
            text: "Outlined"

        MDTextFieldHelperText:
            text: "Helper text"
            mode: "persistent"

        MDTextFieldTrailingIcon:
            icon: "information"

        MDTextFieldMaxLengthText:
            max_text_length: 10
'''


class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)
```

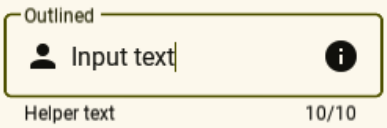
(continues on next page)

(continued from previous page)

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

Declarative Python style

```
from kivymd.uix.textfield import (
    MDTextField,
    MDTextFieldLeadingIcon,
    MDTextFieldHintText,
    MDTextFieldHelperText,
    MDTextFieldTrailingIcon,
    MDTextFieldMaxLengthText,
)

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


class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return MDScreen(
            MDTextField(
                MDTextFieldLeadingIcon(
                    icon="account",
                ),
                MDTextFieldHintText(
                    text="Hint text",
                ),
                MDTextFieldHelperText(
                    text="Helper text",
                    mode="persistent",
                ),
                MDTextFieldTrailingIcon(
                    icon="information",
                ),
                MDTextFieldMaxLengthText(
                    max_text_length=10,
                ),
                mode="outlined",
                size_hint_x=None,
                width="240dp",
                pos_hint={"center_x": 0.5, "center_y": 0.5},
            ),
            md_bg_color=self.theme_cls.backgroundColor,
        )


Example().run()
```



API break

1.2.0 version

```
MDTextField:
    mode: "rectangle"
    hint_text: "Hint text"
    helper_text: "Helper text"
    helper_text_mode: "persistent"
    max_text_length: 10
    icon_right: "information"
```

2.0.0 version

Note: The text field with the *round* type was removed in version *2.0.0*.

```
MDTextField:
    mode: "outlined"

    MDTextFieldLeadingIcon:
        icon: "phone"

    MDTextFieldTrailingIcon:
        icon: "information"

    MDTextFieldHintText:
        text: "Hint text"

    MDTextFieldHelperText:
        text: "Helper text"
        mode: "persistent"

    MDTextFieldMaxLengthText:
        max_text_length: 10
```

API - kivymd.uix.textfield.textfield

class kivymd.uix.textfield.textfield.**AutoFormatTelephoneNumber**

Implements automatic formatting of the text entered in the text field according to the mask, for example '+38 (###) ### ## ##'.

> **Warning:** This class has not yet been implemented and it is not recommended to use it yet.

isnumeric(*value*) → bool

do_backspace(**args*) → None

Do backspace operation from the current cursor position.

field_filter(*value*, *boolean*) → None

format(*value*) → None

class kivymd.uix.textfield.textfield.**Validator**

Container class for various validation methods.

datetime_date

The last valid date as a <class 'datetime.date'> object.

datetime_date is an ObjectProperty and defaults to *None*.

date_interval

The date interval that is valid for input. Can be entered as <class 'datetime.date'> objects or a string format. Both values or just one value can be entered.

In string format, must follow the current date_format. Example: Given date_format -> "mm/dd/yyyy" Input examples -> "12/31/1900", "12/31/2100" or "12/31/1900", None.

date_interval is an ListProperty and defaults to *[None, None]*.

date_format

Format of date strings that will be entered. Available options are: *'dd/mm/yyyy'*, *'mm/dd/yyyy'*, *'yyyy/mm/dd'*.

date_format is an OptionProperty and defaults to *None*.

is_email_valid(*text: str*) → bool

Checks the validity of the email.

is_time_valid(*text: str*) → bool

Checks the validity of the time.

is_date_valid(*text: str*) → bool

Checks the validity of the date.

on_date_interval(**args*) → None

Default event handler for date_interval input.

class kivymd.uix.textfield.textfield.**BaseTextFieldLabel**(**args*, ***kwargs*)

Base texture for *MDTextField* class (helper text, max length, hint text).

For more information, see in the *MDLabel* class documentation.

New in version 2.0.0.

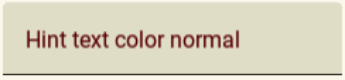
text_color_normal

Text color in (r, g, b, a) or string format when text field is out of focus.

New in version 1.0.0.

Changed in version 2.0.0: The property was moved from class:*~MDTextField* class and renamed from *helper_text_color_normal* to *text_color_normal*.

```
MDTextField:
    mode: "filled"

    MDTextFieldHintText:
        text: "Hint text color normal"
        text_color_normal: "brown"
```



text_color_normal is an ColorProperty and defaults to *None*.

text_color_focus

Text color in (r, g, b, a) or string format when the text field has focus.

New in version 1.0.0.

Changed in version 2.0.0: The property was moved from class:*~MDTextField* class and renamed from *helper_text_color_focus* to *text_color_focus*.

```
MDTextField:

    MDTextFieldHelperText:
        text: "Helper text color focus"
        text_color_focus: "brown"
```

text_color_focus is an ColorProperty and defaults to *None*.

class kivymd.uix.textfield.textfield.**MDTextFieldHelperText**(**args*, ***kwargs*)

Implements the helper text label.

For more information, see in the *BaseTextFieldLabel* class documentation.

New in version 2.0.0.

mode

Helper text mode. Available options are: *'on_error'*, *'persistent'*, *'on_focus'*.

Changed in version 2.0.0: The property was moved from class:*~MDTextField* class and renamed from *helper_text_mode* to *mode*.

On focus

```
MDTextField:
    mode: "filled"

    MDTextFieldHelperText:
        text: "Helper text"
        mode: "on_focus"
```

On error

```
MDTextField:
    mode: "filled"

    MDTextFieldHelperText:
        text: "Helper text"
        mode: "on_error"

    MDTextFieldMaxLengthText:
        max_text_length: 5
```

Persistent

```
MDTextField:
    mode: "filled"

    MDTextFieldHelperText:
        text: "Helper text"
        mode: "persistent"
```

mode is an `OptionProperty` and defaults to *'on_focus'*.

class kivymd.uix.textfield.textfield.**MDTextFieldMaxLengthText**(**args*, ***kwargs*)

Implements the max length text label.

For more information, see in the *BaseTextFieldLabel* class documentation.

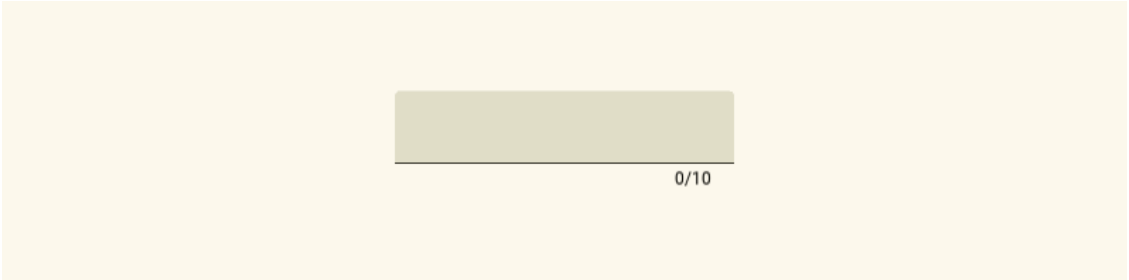
New in version 2.0.0.

max_text_length

Maximum allowed value of characters in a text field.

Changed in version 2.0.0: The property was moved from class:*~MDTextField*.

```
MDTextField:
    mode: "filled"

    MDTextFieldMaxLengthText:
        max_text_length: 10
```



max_text_length is an NumericProperty and defaults to *None*.

class kivymd.uix.textfield.textfield.**MDTextFieldHintText**(**args*, ***kwargs*)

Implements the hint text label.

For more information, see in the *BaseTextFieldLabel* class documentation.

New in version 2.0.0.

```
MDTextField:
    mode: "filled"

    MDTextFieldHintText:
        text: "Hint text"
```

class kivymd.uix.textfield.textfield.**BaseTextFieldIcon**(**args*, ***kwargs*)

Base texture for *MDTextField* class (helper text, max length, hint text).

For more information, see in the *MDIcon* class documentation.

Changed in version 2.0.0.

icon_color_normal

Icon color in (r, g, b, a) or string format when text field is out of focus.

New in version 1.0.0.

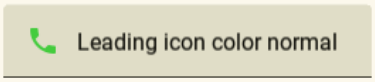
Changed in version 2.0.0: The property was moved from class:*~MDTextField* class and renamed from *icon_right_color_normal/icon_left_color_normal* to *icon_color_normal*.

```
MDTextField:
    mode: "filled"

    MDTextFieldLeadingIcon:
        icon: "phone"
```

(continues on next page)

(continued from previous page)

```
        theme_icon_color: "Custom"
        icon_color_normal: "lightgreen"

    MDTextFieldHintText:
        text: "Leading icon color normal"
```



icon_color_normal is an ColorProperty and defaults to *None*.

icon_color_focus

Icon color in (r, g, b, a) or string format when the text field has focus.

New in version 1.0.0.

Changed in version 2.0.0: The property was moved from class:*~MDTextField* class and renamed from *icon_right_color_focus/icon_left_color_focus `to `icon_color_focus*.

```
MDTextField:
    mode: "filled"

    MDTextFieldLeadingIcon:
        icon: "phone"
        theme_icon_color: "Custom"
        icon_color_focus: "lightgreen"

    MDTextFieldHintText:
        text: "Leading icon color focus"
```

icon_color_focus is an ColorProperty and defaults to *None*.

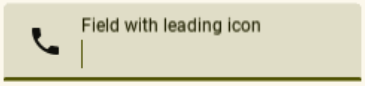
class kivymd.uix.textfield.textfield.**MDTextFieldLeadingIcon**(**args*, ***kwargs*)

Implements the leading icon.

For more information, see in the *BaseTextFieldIcon* class documentation.

New in version 2.0.0.

```
MDTextField:
    mode: "filled"

    MDTextFieldLeadingIcon:
        icon: "phone"

    MDTextFieldHintText:
        text: "Field with leading icon"
```



class kivymd.uix.textfield.textfield.**MDTextFieldTrailingIcon**(**args*, ***kwargs*)

Implements the trailing icon.

For more information, see in the *BaseTextFieldIcon* class documentation.

New in version 2.0.0.

```
MDTextField:
    mode: "filled"

    MDTextFieldTrailingIcon:
        icon: "phone"

    MDTextFieldHintText:
        text: "Field with trailing icon"
```

class kivymd.uix.textfield.textfield.**MDTextField**(**args*, ***kwargs*)

Textfield class.

For more information, see in the *DeclarativeBehavior* and *BackgroundColorBehavior* and *ThemableBehavior* and *TextInput* and *Validator* and *AutoFormatTelephoneNumber* and *StateLayerBehavior* classes documentation.

font_style

Name of the style for the input text.

New in version 2.0.0.

See also:

Font style names

font_style is an StringProperty and defaults to *'Body'*.

role

Role of font style.

New in version 2.0.0.

See also:

Font style roles

role is an StringProperty and defaults to *'large'*.

mode

Text field mode. Available options are: *'outlined'*, *'filled'*.

mode is an OptionProperty and defaults to *'outlined'*.

error_color

Error color in (r, g, b, a) or string format for *required* = *True* or when the text field is in *error* state.

error_color is an ColorProperty and defaults to *None*.

error

If True, then the text field goes into *error* mode.

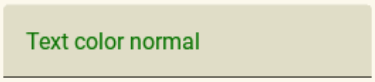
error is an BooleanProperty and defaults to *False*.

text_color_normal

Text color in (r, g, b, a) or string format when text field is out of focus.

New in version 1.0.0.

```
MDTextField:
    theme_text_color: "Custom"
    text_color_normal: "green"
    text: "Text color normal"
```



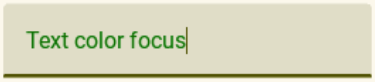
text_color_normal is an ColorProperty and defaults to *None*.

text_color_focus

Text color in (r, g, b, a) or string format when text field has focus.

New in version 1.0.0.

```
MDTextField:
    theme_text_color: "Custom"
    text_color_focus: "green"
    text: "Text color focus"
```



text_color_focus is an ColorProperty and defaults to *None*.

radius

The corner radius for a text field in *filled/outlined* mode.

radius is a VariableListProperty and defaults to *[dp(4), dp(4), 0, 0]*.

required

Required text. If True then the text field requires text.

required is an BooleanProperty and defaults to *False*.

line_color_normal

Line color normal (active indicator) in (r, g, b, a) or string format.

```
MDTextField:
    mode: "filled"
    theme_line_color: "Custom"
    line_color_normal: "green"

    MDTextFieldHelperText:
        text: "Line color normal"
        mode: "persistent"
```

line_color_normal is an ColorProperty and defaults to *None*.

line_color_focus

Line color focus (active indicator) in (r, g, b, a) or string format.

```
MDTextField:
    mode: "filled"
    theme_line_color: "Custom"
```

(continues on next page)

(continued from previous page)

```
    line_color_focus: "green"

    MDTextFieldHelperText:
        text: "Line color focus"
        mode: "persistent"
```

line_color_focus is an ColorProperty and defaults to *None*.

fill_color_normal

Fill background color in (r, g, b, a) or string format in 'fill' mode when] text field is out of focus.

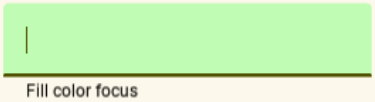
```
MDTextField:
    mode: "filled"
    theme_bg_color: "Custom"
    fill_color_normal: 0, 1, 0, .2

    MDTextFieldHelperText:
        text: "Fill color normal"
        mode: "persistent"
```

fill_color_normal is an ColorProperty and defaults to *None*.

fill_color_focus

Fill background color in (r, g, b, a) or string format in 'fill' mode when the text field has focus.

```
MDTextField:
    mode: "filled"
    theme_bg_color: "Custom"
    fill_color_focus: 0, 1, 0, .2

    MDTextFieldHelperText:
        text: "Fill color focus"
        mode: "persistent"
```



fill_color_focus is an ColorProperty and defaults to *None*.

max_height

Maximum height of the text box when *multiline* = *True*.

```
MDTextField:
    mode: "filled"
    max_height: "200dp"
    multiline: True

    MDTextFieldHelperText:
        text: "multiline=True"
        mode: "persistent"
```

max_height is a NumericProperty and defaults to *0*.

phone_mask

This property has not yet been implemented and it is not recommended to use it yet.

phone_mask is a StringProperty and defaults to ‘’.

validator

The type of text field for entering Email, time, etc. Automatically sets the type of the text field as “error” if the user input does not match any of the set validation types. Available options are: *‘date’*, *‘email’*, *‘time’*.

When using *‘date’*, `date_format` must be defined.

New in version 1.1.0.

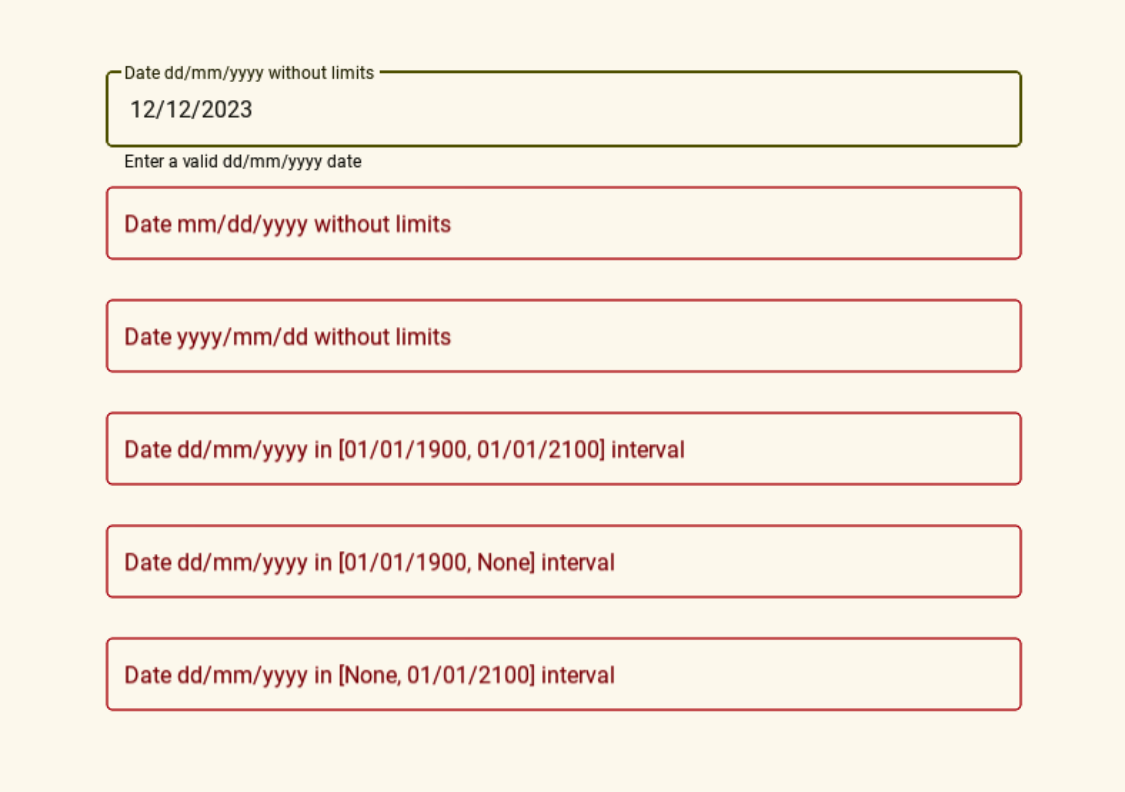
```
MDTextField:
    mode: "filled"
    validator: "email"

    MDTextFieldHintText:
        text: "Email"

    MDTextFieldHelperText:
        text: "user@gmail.com"
        mode: "persistent"
```


```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        orientation: "vertical"
        spacing: "20dp"
        adaptive_height: True
        size_hint_x: .8
        pos_hint: {"center_x": .5, "center_y": .5}

        MDTextField:
            validator: "date"
            date_format: "dd/mm/yyyy"

            MDTextFieldHintText:
                text: "Date dd/mm/yyyy without limits"

            MDTextFieldHelperText:
                text: "Enter a valid dd/mm/yyyy date"

        MDTextField:
            validator: "date"
            date_format: "mm/dd/yyyy"

            MDTextFieldHintText:
                text: "Date mm/dd/yyyy without limits"

            MDTextFieldHelperText:
                text: "Enter a valid mm/dd/yyyy date"

        MDTextField:
            validator: "date"
            date_format: "yyyy/mm/dd"

            MDTextFieldHintText:
                text: "Date yyyy/mm/dd without limits"

            MDTextFieldHelperText:
                text: "Enter a valid yyyy/mm/dd date"
```

(continues on next page)

(continued from previous page)

```
        MDTextField:
            validator: "date"
            date_format: "dd/mm/yyyy"
            date_interval: "01/01/1900", "01/01/2100"

            MDTextFieldHintText:
                text: "Date dd/mm/yyyy in [01/01/1900, 01/01/2100] interval"

            MDTextFieldHelperText:
                text: "Enter a valid dd/mm/yyyy date"

        MDTextField:
            validator: "date"
            date_format: "dd/mm/yyyy"
            date_interval: "01/01/1900", None

            MDTextFieldHintText:
                text: "Date dd/mm/yyyy in [01/01/1900, None] interval"

            MDTextFieldHelperText:
                text: "Enter a valid dd/mm/yyyy date"

        MDTextField:
            validator: "date"
            date_format: "dd/mm/yyyy"
            date_interval: None, "01/01/2100"

            MDTextFieldHintText:
                text: "Date dd/mm/yyyy in [None, 01/01/2100] interval"

            MDTextFieldHelperText:
                text: "Enter a valid dd/mm/yyyy date"
'''


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


Example().run()
```



validator is an OptionProperty and defaults to *None*.

update_colors(*theme_manager:* kivymd.theming.ThemeManager, *theme_color: str*) → None

Fired when the *primary_palette* or *theme_style* value changes.

add_widget(*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)
```

set_texture_color(*texture*, *canvas_group*, *color: list*, *error: bool = False*) → None

Animates the color of the leading/trailing icons/hint/helper/max length text.

set_pos_hint_text(*y: float*, *x: float*) → None

Animates the x-axis width and y-axis height of the hint text.

set_hint_text_font_size() → None

Animates the font size of the hint text.

set_space_in_line(*left_width: float | int*, *right_width: float | int*) → None

Animates the length of the right line of the text field for the hint text.

set_max_text_length() → None

Fired when text is entered into a text field. Set max length text and updated max length texture.

set_text(*instance*, *text: str*) → None

Fired when text is entered into a text field.

on_focus(*instance*, *focus: bool*) → None

Fired when the *focus* value changes.

on_disabled(*instance*, *disabled: bool*) → None

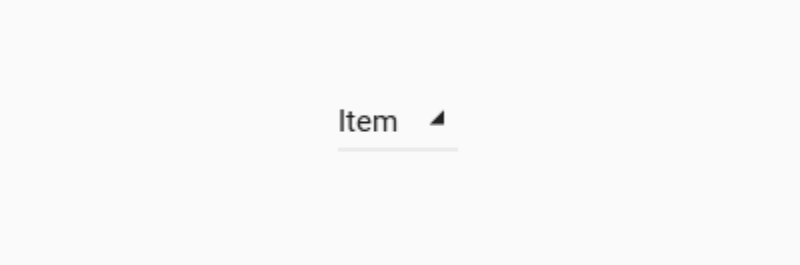
Fired when the *disabled* value changes.

on_error(*instance*, *error: bool*) → None

Changes the primary colors of the text box to match the *error* value (text field is in an error state or not).

on_height(*instance*, *value_height: float*) → None

2.3.31 DropdownItem



Usage

```
from kivy.lang import Builder
from kivymd.uix.menu import MDDropdownMenu

from kivymd.app import MDApp

KV = '''
MDScreen
    md_bg_color: self.theme_cls.backgroundColor

    MDDropDownItem:
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.open_menu(self)

        MDDropDownItemText:
            id: drop_text
            text: "Item"
'''


class Example(MDApp):
    def open_menu(self, item):
        menu_items = [
            {
                "text": f"{i}",
                "on_release": lambda x=f"Item {i}": self.menu_callback(x),
            } for i in range(5)
        ]
        MDDropdownMenu(caller=item, items=menu_items).open()

    def menu_callback(self, text_item):
        self.root.ids.drop_text.text = text_item

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

See also:

Work with the class MDDropdownMenu see here

API break

1.2.0 version

```
MDDropDownItem:
    text: 'Item'
    on_release: print(*args)
```

2.0.0 version

```
MDDropDownItem:
    on_release:  print(*args)

    MDDropDownItemText:
        text: "Item text"
```

API - kivymd.uix.dropdownitem.dropdownitem

class kivymd.uix.dropdownitem.dropdownitem.**MDDropDownItemText**(**args*, ***kwargs*)

Base texture for *MDDropDownItem* class (item text).

For more information, see in the *MDLabel* class documentation.

New in version 2.0.0.

class kivymd.uix.dropdownitem.dropdownitem.**MDDropDownItem**(**args*, ***kwargs*)

Dropdown item class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and ButtonBehavior and BoxLayout classes documentation.

add_widget(*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)
```

update_text_item(*instance*, *value*) → None

Updates the text of the item.

on_disabled(*instance*, *value*) → None

Fired when the values of `disabled` change.

on__drop_down_text(*instance*, *value*) → None

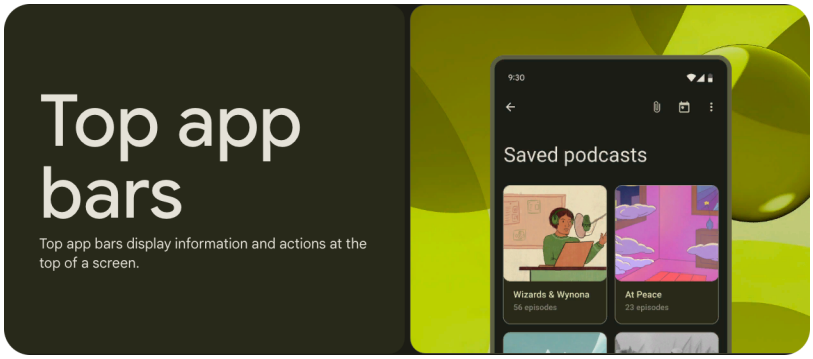
Fired when the values of `_drop_down_text` change.

2.3.32 Appbar

See also:

Material Design spec, App bars: top

Material Design spec, App bars: bottom

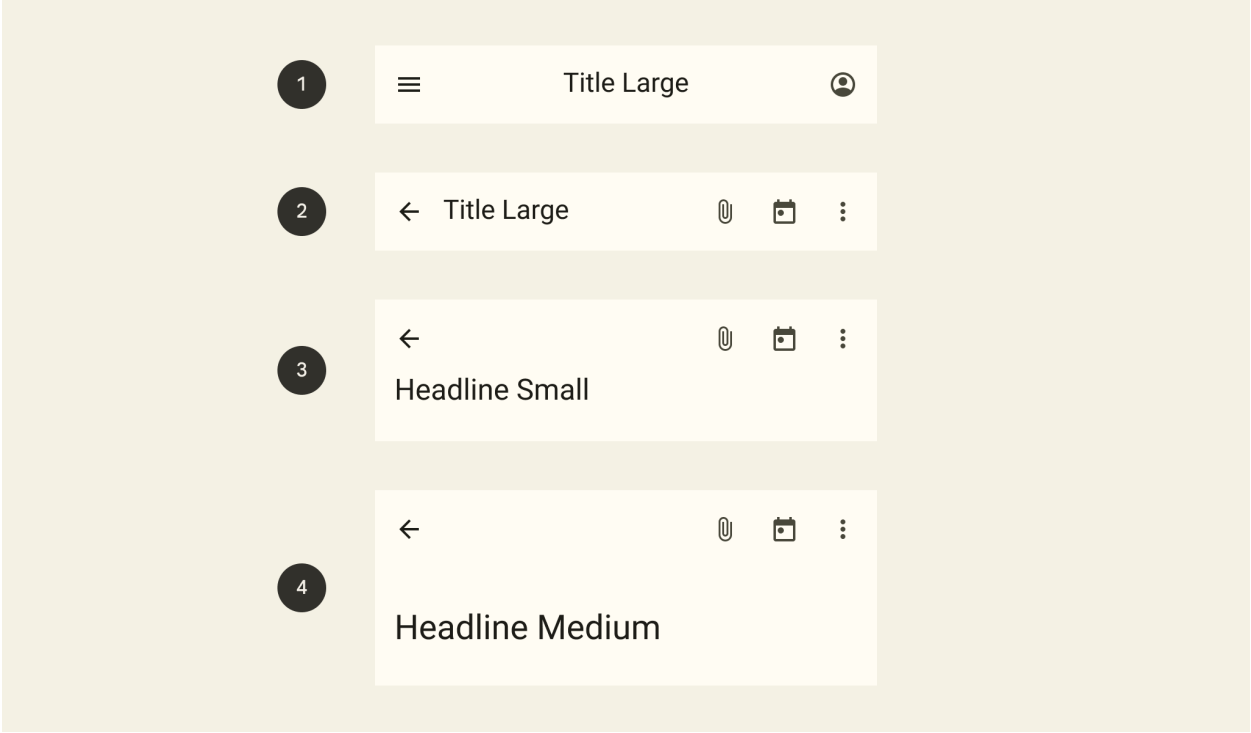


KivyMD provides the following bar positions for use:

- *TopAppBar*
- *BottomAppBar*

TopAppBar

- Contains a title and actions related to the current screen
- Four types: center-aligned, small, medium, and large
- On scroll, apply a container fill color to separate app bar from body content
- Top app bars have the same width as the device window



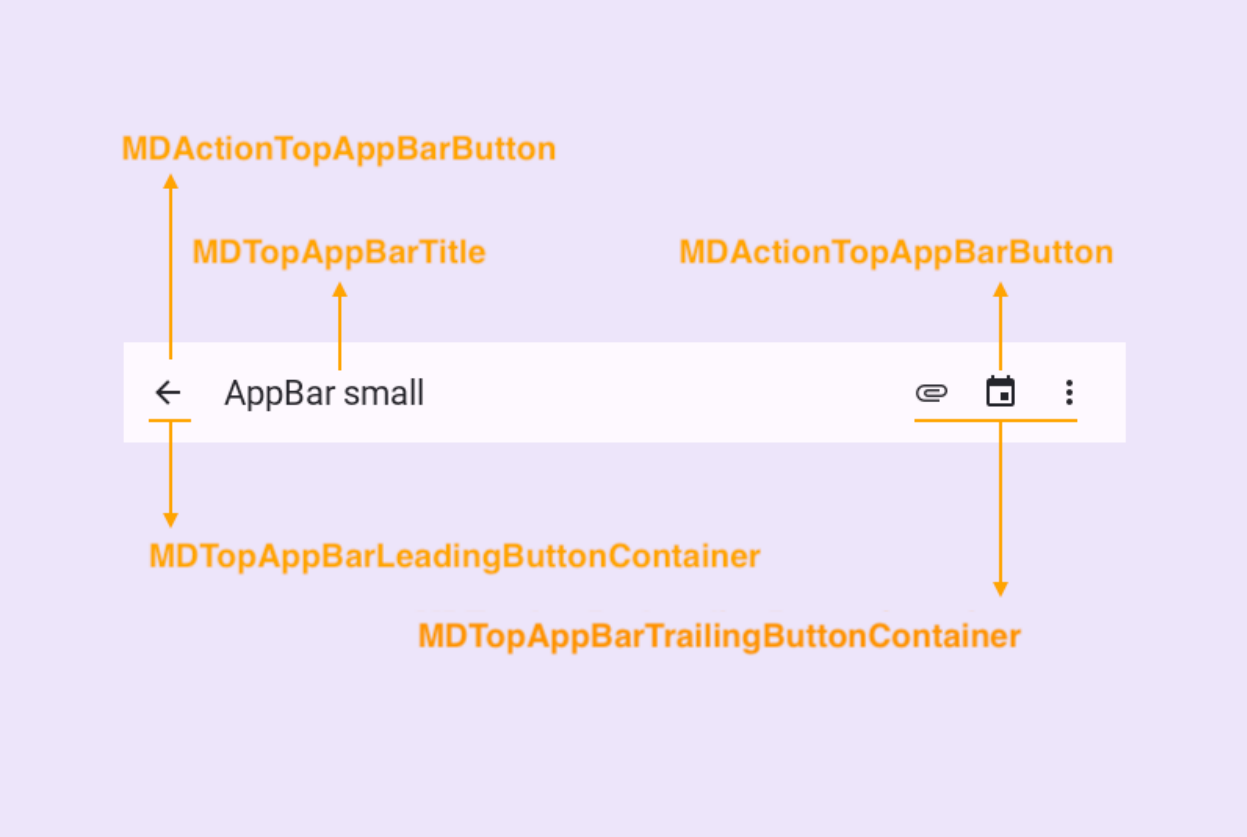
1. Center-aligned
2. Small
3. Medium
4. Large

Note: KivyMD does not provide a *Center-aligned* type panel. But you can easily create this pit panel yourself (read the documentation below).

Usage

```
MDTopAppBar:
    type: "small"

    MDTopAppBarLeadingButtonContainer:

        MDActionTopAppBarButton:
            icon: "menu"

    MDTopAppBarTitle:
        text: "AppBar Center-aligned"
        pos_hint: {"center_x": .5}

    MDTopAppBarTrailingButtonContainer:

        MDActionTopAppBarButton:
            icon: "account-circle-outline"
```

Anatomy



Configurations

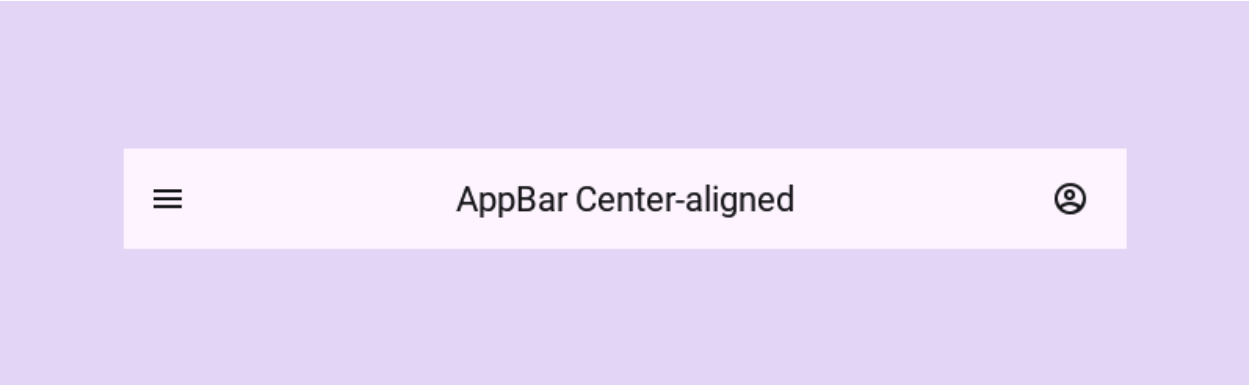
1. Center-aligned

```
MDScreen:
    md_bg_color: self.theme_cls.secondaryContainerColor

    MDTopAppBar:
        type: "small"
        size_hint_x: .8
        pos_hint: {"center_x": .5, "center_y": .5}

        MDTopAppBarLeadingButtonContainer:

            MDActionTopAppBarButton:
                icon: "menu"

        MDTopAppBarTitle:
            text: "AppBar small"
            pos_hint: {"center_x": .5}

        MDTopAppBarTrailingButtonContainer:
```

(continues on next page)

(continued from previous page)

```
            MDActionTopAppBarButton:
                icon: "account-circle-outline"
```



2. Small

```
MDScreen:
    md_bg_color: self.theme_cls.secondaryContainerColor

    MDTopAppBar:
        type: "small"
        size_hint_x: .8
        pos_hint: {"center_x": .5, "center_y": .5}

        MDTopAppBarLeadingButtonContainer:

            MDActionTopAppBarButton:
                icon: "arrow-left"

        MDTopAppBarTitle:
            text: "AppBar small"

        MDTopAppBarTrailingButtonContainer:

            MDActionTopAppBarButton:
                icon: "attachment"

            MDActionTopAppBarButton:
                icon: "calendar"

            MDActionTopAppBarButton:
                icon: "dots-vertical"
```

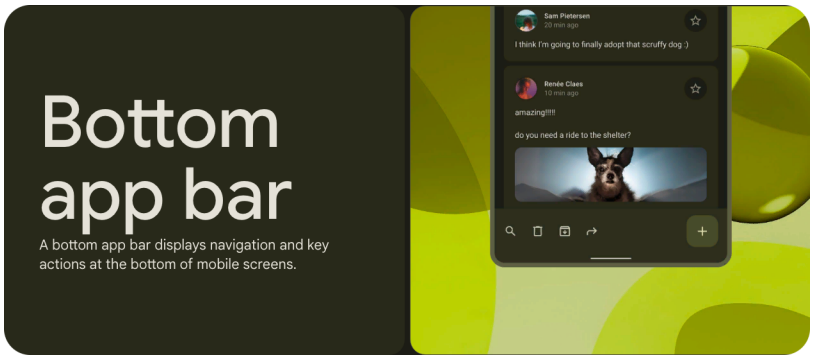
3. Medium

```
MDTopAppBar:
    type: "medium"
```

4. Large

```
MDTopAppBar:
    type: "large"
```

BottomAppBar




```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDBottomAppBar:

        MDFabBottomAppBarButton:
            icon: "plus"
'''


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


Example().run()
```

Add action items

```
#:import MDActionBottomAppBarButton kivymd.uix.appbar.MDActionBottomAppBarButton


MDScreen:

    MDBottomAppBar:
        action_items:
            [
            MDActionBottomAppBarButton(icon="gmail"),
            MDActionBottomAppBarButton(icon="label-outline"),
            MDActionBottomAppBarButton(icon="bookmark"),
            ]
```

Change action items

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.appbar import MDActionBottomAppBarButton

KV = '''
#:import MDActionBottomAppBarButton kivymd.uix.appbar.MDActionBottomAppBarButton


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDBottomAppBar:
        id: bottom_appbar
        action_items:
            [
            MDActionBottomAppBarButton(icon="gmail"),
            MDActionBottomAppBarButton(icon="bookmark"),
            ]

        MDFabBottomAppBarButton:
            icon: "plus"
            on_release: app.change_actions_items()
```

(continues on next page)

(continued from previous page)

```
'''


class Example(MDApp):
    def change_actions_items(self):
        self.root.ids.bottom_appbar.action_items = [
            MDActionBottomAppBarButton(icon="magnify"),
            MDActionBottomAppBarButton(icon="trash-can-outline"),
            MDActionBottomAppBarButton(icon="download-box-outline"),
        ]

    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)


Example().run()
```

A practical example

```
import asynckivy

from kivy.clock import Clock
from kivy.lang import Builder
from kivy.properties import StringProperty, BooleanProperty, ObjectProperty
from kivy.uix.behaviors import FocusBehavior
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.uix.recycleview.layout import LayoutSelectionBehavior
from kivy.uix.recycleview.views import RecycleDataViewBehavior

from kivymd.uix.appbar import MDActionBottomAppBarButton
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.app import MDApp

from faker import Faker  # pip install Faker

KV = '''
#:import MDFabBottomAppBarButton kivymd.uix.appbar.MDFabBottomAppBarButton


<UserCard>
    orientation: "vertical"
    adaptive_height: True
    md_bg_color: "#373A22" if self.selected else "#1F1E15"
    radius: 16
    padding: 0, 0, 0, "16dp"

    MDListItem:
        theme_bg_color: "Custom"
```

(continues on next page)

(continued from previous page)

```
        md_bg_color: root.md_bg_color
        radius: root.radius
        ripple_effect: False

        MDListItemLeadingAvatar:
            source: root.avatar
            # radius: self.height / 2

        MDListItemHeadlineText:
            text: root.name
            theme_text_color: "Custom"
            text_color: "#8A8D79"

        MDListItemSupportingText:
            text: root.time
            theme_text_color: "Custom"
            text_color: "#8A8D79"

    MDLabel:
        text: root.text
        adaptive_height: True
        theme_text_color: "Custom"
        text_color: "#8A8D79"
        padding_x: "16dp"
        shorten: True
        shorten_from: "right"

    Widget:


MDFloatLayout:
    md_bg_color: "#151511"

    RecycleView:
        id: card_list
        viewclass: "UserCard"

        SelectableRecycleGridLayout:
            orientation: 'vertical'
            spacing: "16dp"
            padding: "16dp"
            default_size: None, dp(120)
            default_size_hint: 1, None
            size_hint_y: None
            height: self.minimum_height
            multiselect: True
            touch_multiselect: True

    MDBottomAppBar:
        id: bottom_appbar
        scroll_cls: card_list
        allow_hidden: True
```

(continues on next page)

(continued from previous page)

```
        theme_bg_color: "Custom"
        md_bg_color: "#232217"

        MDFabBottomAppBarButton:
            id: fab_button
            icon: "plus"
            theme_bg_color: "Custom"
            md_bg_color: "#373A22"
            theme_icon_color: "Custom"
            icon_color: "#ffffff"
'''


class UserCard(RecycleDataViewBehavior, MDBoxLayout):
    name = StringProperty()
    time = StringProperty()
    text = StringProperty()
    avatar = StringProperty()
    callback = ObjectProperty(lambda x: x)

    index = None
    selected = BooleanProperty(False)
    selectable = BooleanProperty(True)

    def refresh_view_attrs(self, rv, index, data):
        self.index = index
        return super().refresh_view_attrs(rv, index, data)

    def on_touch_down(self, touch):
        if super().on_touch_down(touch):
            return True
        if self.collide_point(*touch.pos) and self.selectable:
            Clock.schedule_once(self.callback)
            return self.parent.select_with_touch(self.index, touch)

    def apply_selection(self, rv, index, is_selected):
        self.selected = is_selected
        rv.data[index]["selected"] = is_selected


class SelectableRecycleGridLayout(
    FocusBehavior, LayoutSelectionBehavior, RecycleBoxLayout
):
    pass


class BottomAppBarButton(MDActionBottomAppBarButton):
    theme_icon_color = "Custom"
    icon_color = "#8A8D79"


class Example(MDApp):
```

(continues on next page)

(continued from previous page)

```
    selected_cards = False

    def build(self):
        return Builder.load_string(KV)

    def on_tap_card(self, *args):
        datas = [data["selected"] for data in self.root.ids.card_list.data]
        if True in datas and not self.selected_cards:
            self.root.ids.bottom_appbar.action_items = [
                BottomAppBarButton(icon="gmail"),
                BottomAppBarButton(icon="label-outline"),
                BottomAppBarButton(icon="bookmark"),
            ]
            self.root.ids.fab_button.icon = "pencil"
            self.selected_cards = True
        else:
            if len(list(set(datas))) == 1 and not list(set(datas))[0]:
                self.selected_cards = False
            if not self.selected_cards:
                self.root.ids.bottom_appbar.action_items = [
                    BottomAppBarButton(icon="magnify"),
                    BottomAppBarButton(icon="trash-can-outline"),
                    BottomAppBarButton(icon="download-box-outline"),
                ]
                self.root.ids.fab_button.icon = "plus"

    def on_start(self):
        async def generate_card():
            for i in range(10):
                await asynckivy.sleep(0)
                self.root.ids.card_list.data.append(
                    {
                        "name": fake.name(),
                        "time": fake.date(),
                        "avatar": fake.image_url(),
                        "text": fake.text(),
                        "selected": False,
                        "callback": self.on_tap_card,
                    }
                )

        self.on_tap_card()
        fake = Faker()
        Clock.schedule_once(lambda x: asynckivy.start(generate_card()))


Example().run()
```

API break

1.2.0 version

```
MDTopAppBar:
    type_height: "large"
    headline_text: "Headline"
    left_action_items: [["arrow-left", lambda x: x]]
    right_action_items:
        [             ["attachment", lambda x: x],             ["calendar", lambda x: x],
↪             ["dots-vertical", lambda x: x],             ]
    anchor_title: "left"
```

2.0.0 version

```
MDTopAppBar:
    type: "large"

    MDTopAppBarLeadingButtonContainer:

        MDActionTopAppBarButton:
            icon: "arrow-left"

    MDTopAppBarTitle:
        text: "AppBar small"

    MDTopAppBarTrailingButtonContainer:

        MDActionTopAppBarButton:
            icon: "attachment"

        MDActionTopAppBarButton:
            icon: "calendar"

        MDActionTopAppBarButton:
            icon: "dots-vertical"
```

API - kivymd.uix.appbar.appbar

class kivymd.uix.appbar.appbar.**MDFabBottomAppBarButton**(***kwargs*)

Implements a floating action button (FAB) for a bar with type 'bottom'.

For more information, see in the *MDFabButton* and *RotateBehavior* and *ScaleBehavior* and classes documentation.

class kivymd.uix.appbar.appbar.**MDActionTopAppBarButton**(***kwargs*)

Implements action buttons on the bar.

For more information, see in the *MDIconButton* class documentation.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the button when the button is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

class kivymd.uix.appbar.appbar.**MDActionBottomAppBarButton**(***kwargs*)

Implements action buttons for a :class:'~kivymd.uix.appbar.appbar.MDBottomAppBar' class.

New in version 1.2.0.

For more information, see in the *MDActionTopAppBarButton* class documentation.

class kivymd.uix.appbar.appbar.**MDTopAppBarTitle**(**args*, ***kwargs*)

Implements the panel title.

New in version 2.0.0.

For more information, see in the *MDLabel* class documentation.

on_text(*instance*, *value*) → None

Fired when the `text` value changes.

on_pos_hint(*instance*, *value*) → None

Fired when the `pos_hint` value changes.

class kivymd.uix.appbar.appbar.**MDTopAppBarLeadingButtonContainer**(**args*, ***kwargs*)

Implements a container for the leading action buttons.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

class kivymd.uix.appbar.appbar.**MDTopAppBarTrailingButtonContainer**(**args*, ***kwargs*)

Implements a container for the trailing action buttons.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

class kivymd.uix.appbar.appbar.**MDTopAppBar**(**args*, ***kwargs*)

Top app bar class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *CommonElevationBehavior* and *BackgroundColorBehavior* and BoxLayout and *WindowController* classes documentation.

Events

on_action_button
Method for the button used for the *MDBottomAppBar* class.

set_bars_color

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

New in version 1.0.0.

See set_bars_colors for more information.

set_bars_color is an BooleanProperty and defaults to *False*.

type

Bar height type.

New in version 1.0.0.

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

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

on_type(*instance*, *value*) → None

on_size(*instance*, *size*) → None

Fired when the application screen size changes.

add_widget(*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)
```

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

Bottom app bar class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and *CommonElevationBehavior* and `FloatLayout` classes documentation.

Events

on_show_bar
The method is fired when the *MDBottomAppBar* panel is shown.

on_hide_bar
The method is fired when the *MDBottomAppBar* panel is hidden.

action_items

The icons on the left bar.

New in version 1.2.0.

action_items is an ListProperty and defaults to *[]*.

animation

TODO: add description. # FIXME: changing the value does not affect anything.

New in version 1.2.0.

animation is an BooleanProperty and defaults to *True*.

show_transition

Type of button display transition.

New in version 1.2.0.

show_transition is a StringProperty and defaults to *'linear'*.

hide_transition

Type of button hidden transition.

New in version 1.2.0.

hide_transition is a StringProperty and defaults to *'in_back'*.

hide_duration

Duration of button hidden transition.

New in version 1.2.0.

hide_duration is a NumericProperty and defaults to *0.2*.

show_duration

Duration of button display transition.

New in version 1.2.0.

show_duration is a NumericProperty and defaults to *0.2*.

scroll_cls

Widget inherited from the ScrollView class. The value must be set if the *allow_hidden* parameter is *True*.

New in version 1.2.0.

scroll_cls is a ObjectProperty and defaults to *None*.

allow_hidden

Allows or disables hiding the panel when scrolling content. If the value is *True*, the *scroll_cls* parameter must be specified.

New in version 1.2.0.

allow_hidden is a BooleanProperty and defaults to *False*.

bar_is_hidden

Is the panel currently hidden.

New in version 1.2.0.

bar_is_hidden is a BooleanProperty and defaults to *False*.

button_centering_animation(*button:* MDActionBottomAppBarButton | MDFabBottomAppBarButton) → None

Animation of centering buttons for MDActionOverFlowButton, *MDActionBottomAppBarButton* and *MDFabBottomAppBarButton* classes.

check_scroll_direction(*scroll_cls*, *y: float*) → None

Checks the scrolling direction. Depending on the scrolling direction, hides or shows the *MDBottomAppBar* panel.

show_bar() → None

Show *MDBottomAppBar* panel.

hide_bar() → None

Hide *MDBottomAppBar* panel.

on_show_bar(**args*) → None

The method is fired when the *MDBottomAppBar* panel is shown.

on_hide_bar(**args*) → None

The method is fired when the *MDBottomAppBar* panel is hidden.

on_scroll_cls(*instance*, *scroll_cls*) → None

Fired when the value of the *scroll_cls* attribute changes.

on_size(**args*) → None

Fired when the root screen is resized.

on_action_items(*instance*, *value: list*) → None

Fired when the value of the *action_items* attribute changes.

set_fab_opacity(**ars*) → None

Sets the transparency value of the:class:*~MDFabBottomAppBarButton* button.

set_fab_icon(*instance*, *value*) → None

Animates the size of the *MDFabBottomAppBarButton* button.

add_widget(*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.

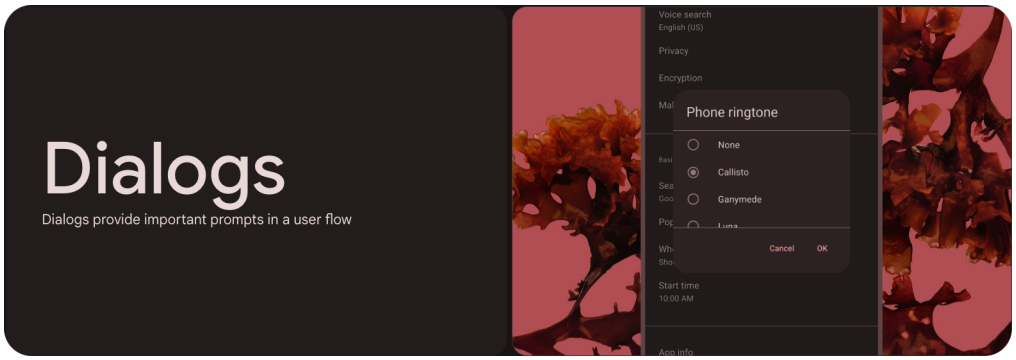
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.33 Dialog

See also:

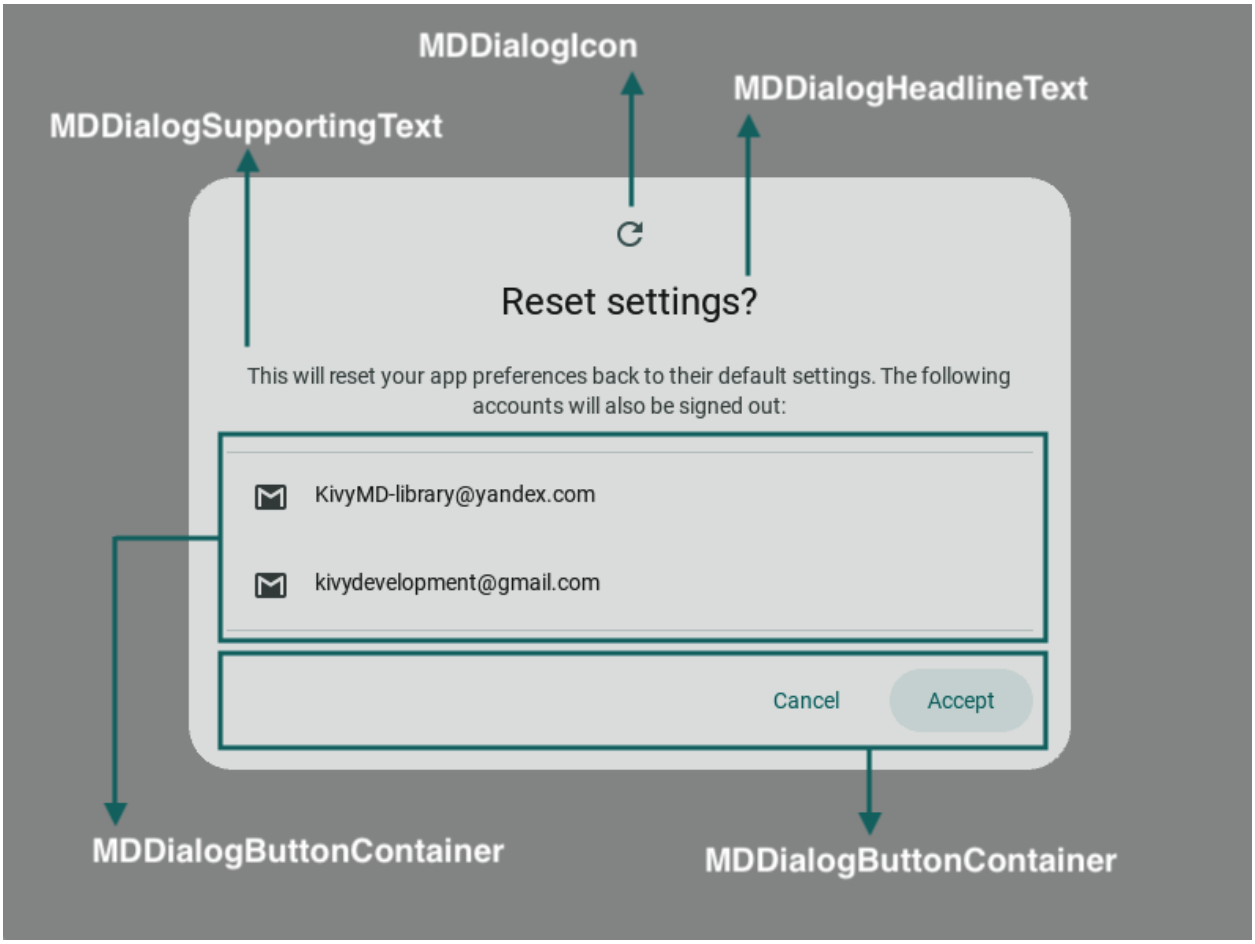
Material Design spec, Dialogs

Dialogs provide important prompts in a user flow.



- Use dialogs to make sure users act on information
- Two types: basic and full-screen (full-screen not provided in KivyMD)
- Should be dedicated to completing a single task
- Can also display information relevant to the task
- Commonly used to confirm high-risk actions like deleting progress

Anatomy



Example

```
from kivy.lang import Builder
from kivy.uix.widget import Widget

from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.dialog import (
    MDDialog,
    MDDialogIcon,
    MDDialogHeadlineText,
    MDDialogSupportingText,
    MDDialogButtonContainer,
    MDDialogContentContainer,
)
from kivymd.uix.divider import MDDivider
from kivymd.uix.list import (
    MDListItem,
    MDListItemLeadingIcon,
```

(continues on next page)

(continued from previous page)

```
    MDListItemSupportingText,
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_alert_dialog()

        MDButtonText:
            text: "Show dialog"
'''


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

    def show_alert_dialog(self):
        MDDialog(
            # ----------------------------Icon----------------------------
            MDDialogIcon(
                icon="refresh",
            ),
            # -----------------------Headline text-------------------------
            MDDialogHeadlineText(
                text="Reset settings?",
            ),
            # -----------------------Supporting text-----------------------
            MDDialogSupportingText(
                text="This will reset your app preferences back to their "
                "default settings. The following accounts will also "
                "be signed out:",
            ),
            # -----------------------Custom content------------------------
            MDDialogContentContainer(
                MDDivider(),
                MDListItem(
                    MDListItemLeadingIcon(
                        icon="gmail",
                    ),
                    MDListItemSupportingText(
                        text="KivyMD-library@yandex.com",
                    ),
                    theme_bg_color="Custom",
                    md_bg_color=self.theme_cls.transparentColor,
                ),
                MDListItem(
                    MDListItemLeadingIcon(
                        icon="gmail",
```

(continues on next page)

(continued from previous page)

```
                    ),
                    MDListItemSupportingText(
                        text="kivydevelopment@gmail.com",
                    ),
                    theme_bg_color="Custom",
                    md_bg_color=self.theme_cls.transparentColor,
                ),
                MDDivider(),
                orientation="vertical",
            ),
            # --------------------Button container-----------------------
            MDDialogButtonContainer(
                Widget(),
                MDButton(
                    MDButtonText(text="Cancel"),
                    style="text",
                ),
                MDButton(
                    MDButtonText(text="Accept"),
                    style="text",
                ),
                spacing="8dp",
            ),
            # ------------------------------------------------------------
        ).open()


Example().run()
```

Warning: Do not try to use the MDDialog widget in KV files.

API break

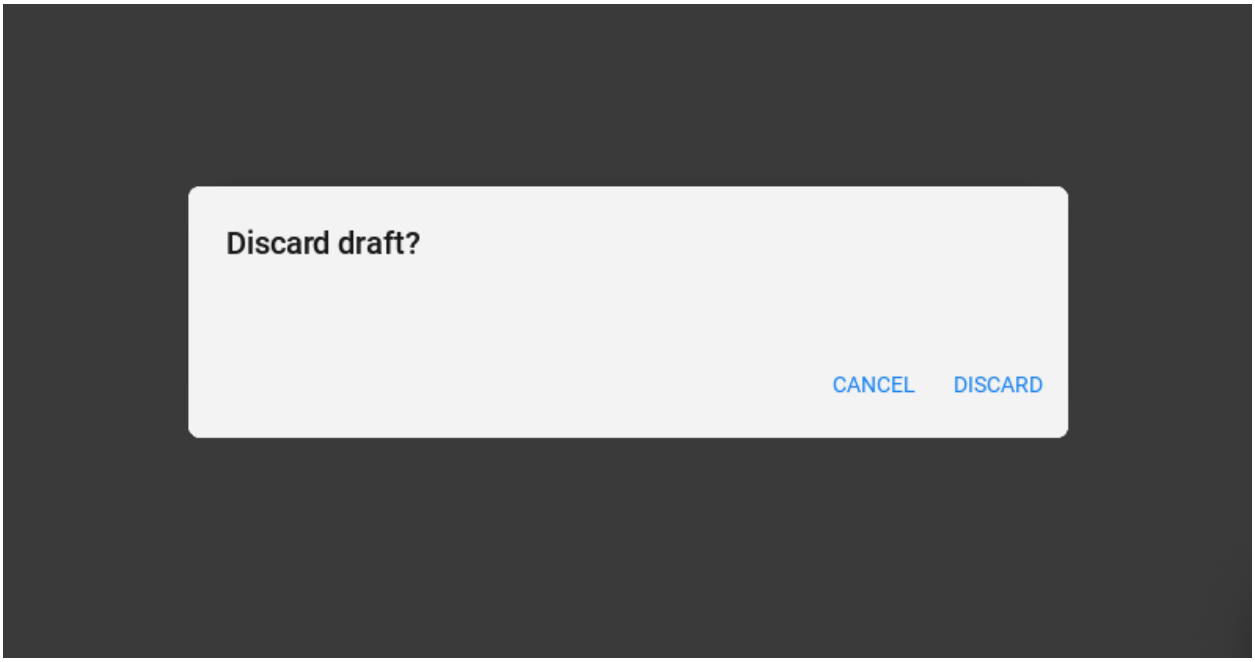
1.2.0 version

```
from kivy.uix.widget import Widget

from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.dialog import MDDialog


class Example(MDApp):
    def build(self):
        return Widget()

    def on_start(self):
```

(continues on next page)

(continued from previous page)

```
        MDDialog(
            title="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,
                ),
            ],
        ).open()


Example().run()
```

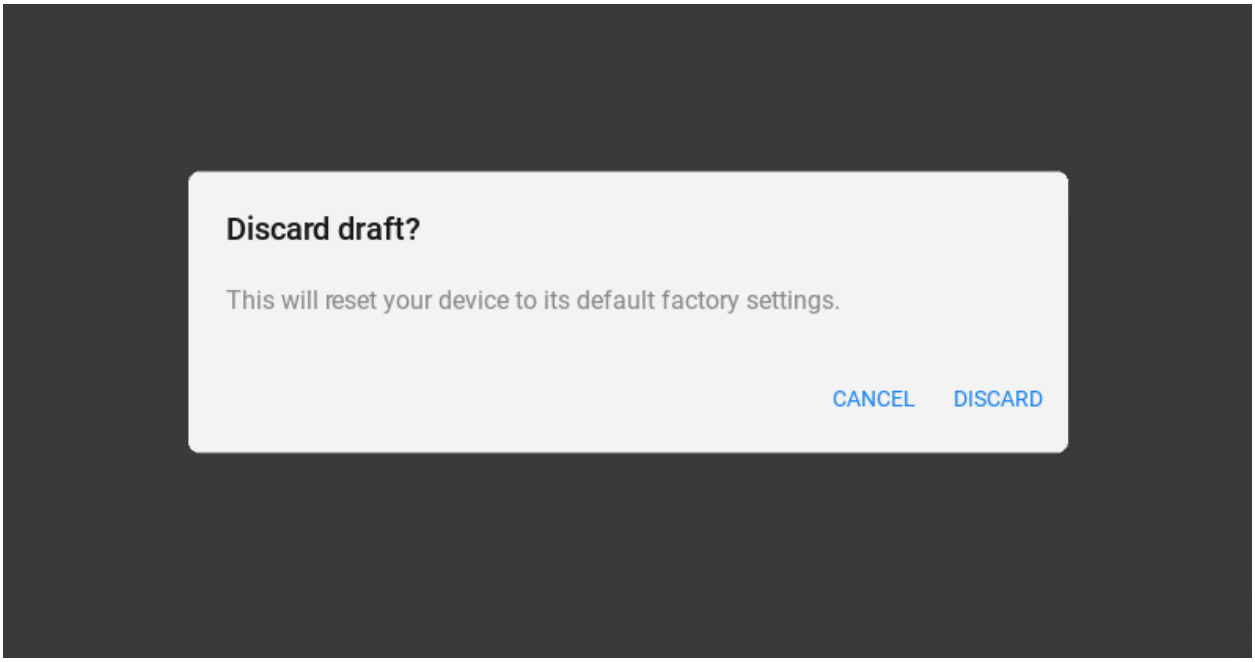



```
from kivy.uix.widget import Widget

from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.dialog import MDDialog


class Example(MDApp):
    def build(self):
        return Widget()

    def on_start(self):
```

(continues on next page)

(continued from previous page)

```
        MDDialog(
            title="Discard draft?",
            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="DISCARD",
                    theme_text_color="Custom",
                    text_color=self.theme_cls.primary_color,
                ),
            ],
        ).open()


Example().run()
```

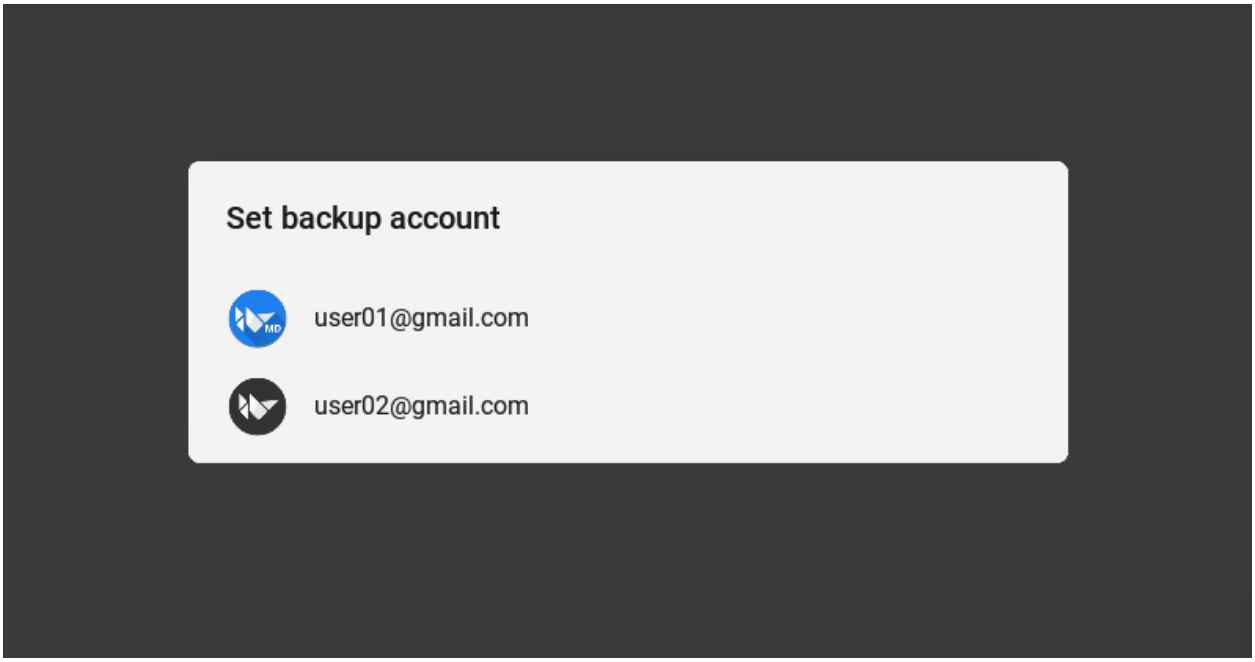



```
from kivy.lang import Builder
from kivy.properties import StringProperty
from kivy.uix.widget import Widget

from kivymd import images_path
from kivymd.app import MDApp
from kivymd.uix.dialog import MDDialog
from kivymd.uix.list import OneLineAvatarListItem

KV = '''
<Item>
```

(continues on next page)

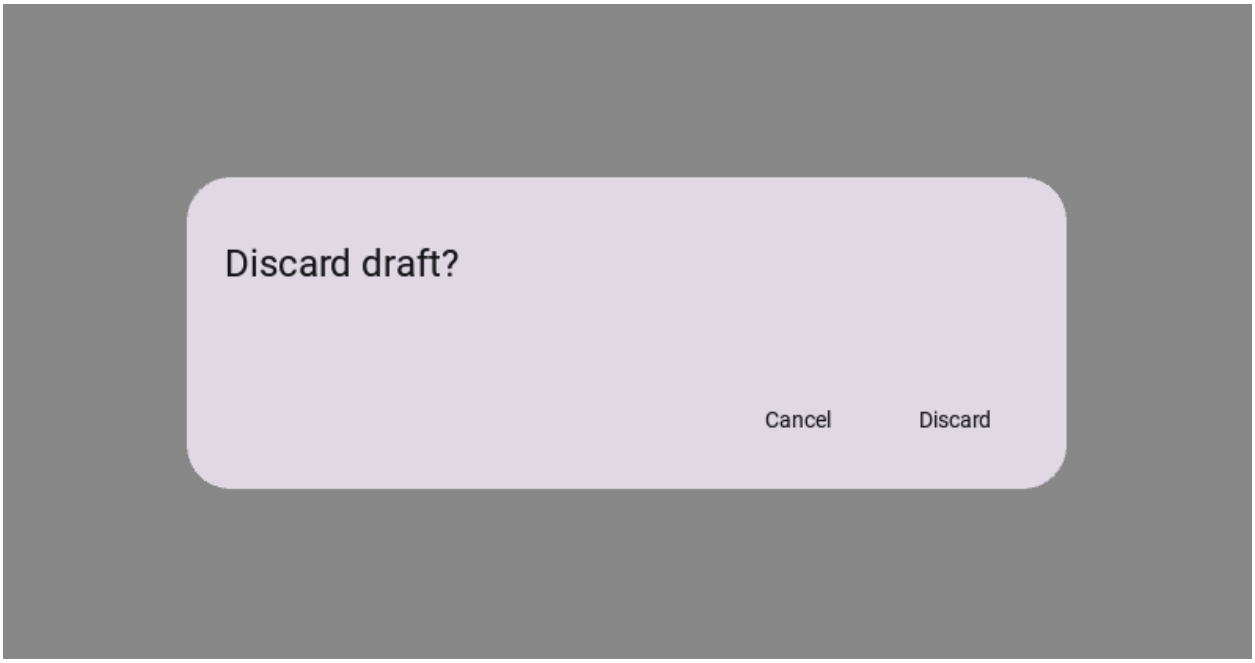
(continued from previous page)

```
    ImageLeftWidget:
        source: root.source
'''


class Item(OneLineAvatarListItem):
    divider = None
    source = StringProperty()


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

    def on_start(self):
        MDDialog(
            title="Set backup account",
            type="simple",
            items=[
                Item(text="user01@gmail.com", source=f"{images_path}/logo/kivymd-icon-
↪128.png"),
                Item(text="user02@gmail.com", source="data/logo/kivy-icon-128.png"),
            ],
        ).open()


Example().run()
```



2.2.0 version

```
from kivy.uix.widget import Widget

from kivymd.uix.widget import MDWidget
from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.dialog import MDDialog, MDDialogHeadlineText, MDDialogButtonContainer


class Example(MDApp):
    def build(self):
        return MDWidget(md_bg_color=self.theme_cls.backgroundColor)

    def on_start(self):
        MDDialog(
            MDDialogHeadlineText(
                text="Discard draft?",
                halign="left",
            ),
            MDDialogButtonContainer(
                Widget(),
                MDButton(
                    MDButtonText(text="Cancel"),
                    style="text",
                ),
                MDButton(
                    MDButtonText(text="Discard"),
                    style="text",
                ),
                spacing="8dp",
            ),
        ).open()


Example().run()
```

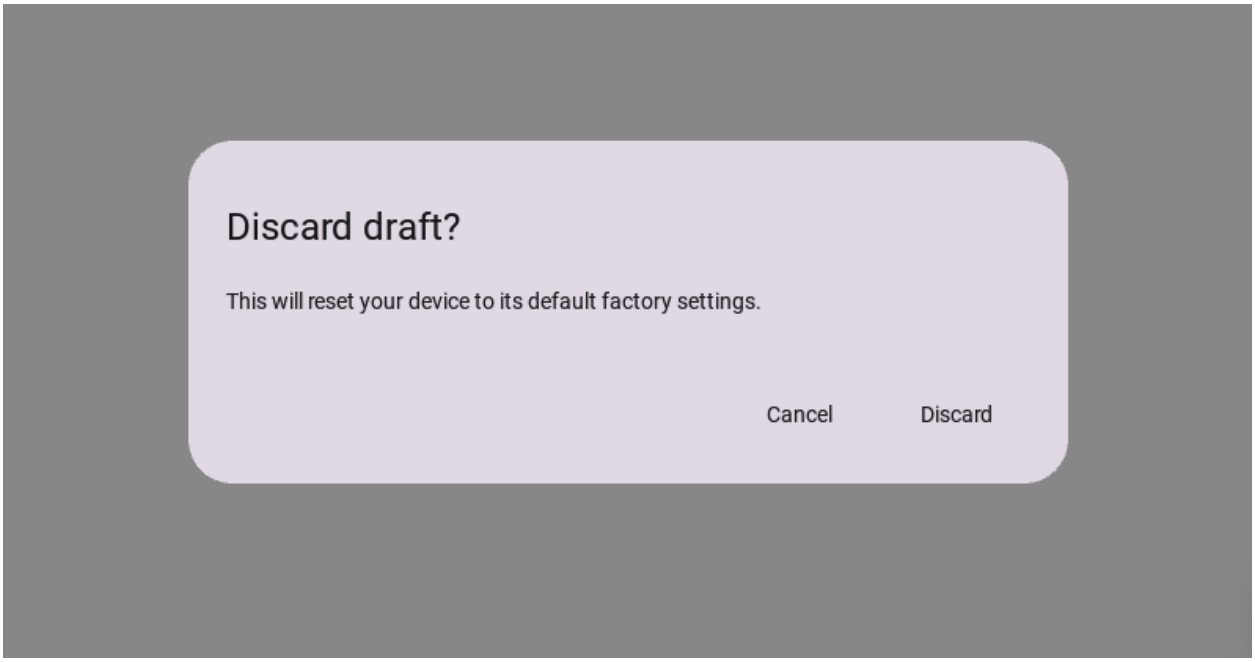



```
from kivy.uix.widget import Widget

from kivymd.uix.widget import MDWidget
from kivymd.app import MDApp
from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.dialog import MDDialog, MDDialogHeadlineText, MDDialogButtonContainer


class Example(MDApp):
    def build(self):
        return MDWidget(md_bg_color=self.theme_cls.backgroundColor)

    def on_start(self):
        MDDialog(
            MDDialogHeadlineText(
                text="Discard draft?",
                halign="left",
            ),
            MDDialogSupportingText(
                text="This will reset your device to its default factory settings.",
                halign="left",
            ),
            MDDialogButtonContainer(
                Widget(),
                MDButton(
                    MDButtonText(text="Cancel"),
                    style="text",
                ),
                MDButton(
                    MDButtonText(text="Discard"),
                    style="text",
                ),
```

(continues on next page)

(continued from previous page)

```
                spacing="8dp",
            ),
        ).open()


Example().run()
```

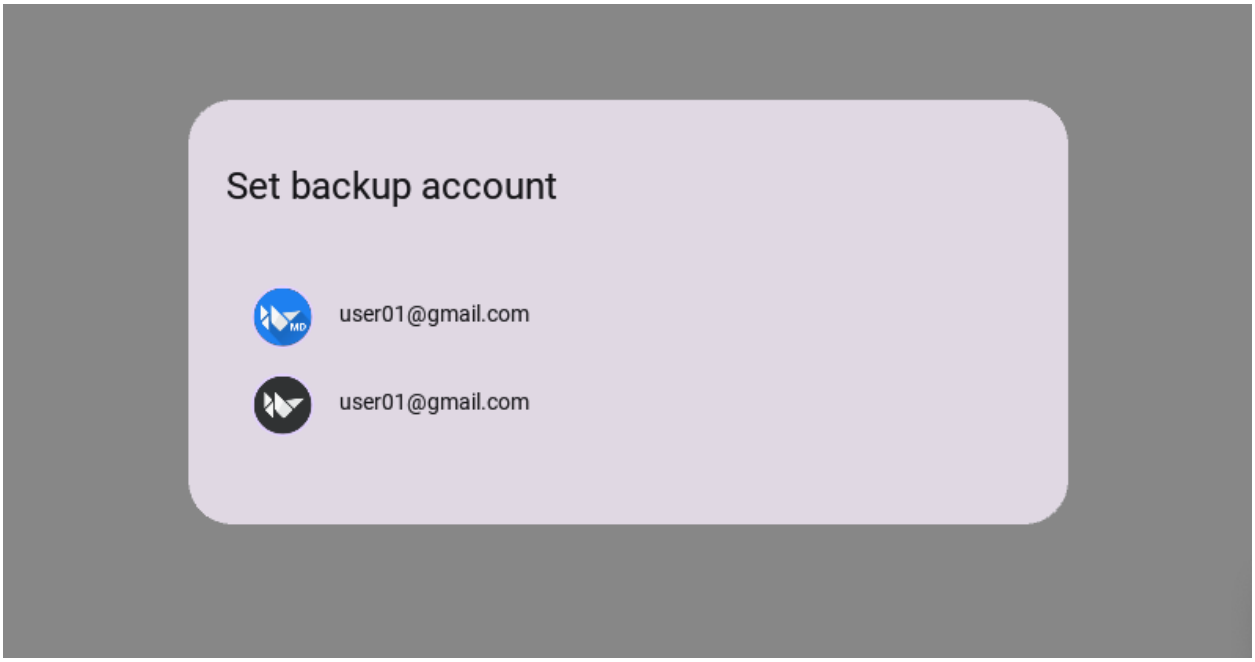



```
from kivymd import images_path
from kivymd.uix.widget import MDWidget
from kivymd.app import MDApp
from kivymd.uix.dialog import (
    MDDialog,
    MDDialogHeadlineText,
    MDDialogContentContainer,
)
from kivymd.uix.list import (
    MDListItem,
    MDListItemLeadingAvatar,
    MDListItemSupportingText,
)


class Example(MDApp):
    def build(self):
        return MDWidget(md_bg_color=self.theme_cls.backgroundColor)

    def on_start(self):
        MDDialog(
            MDDialogHeadlineText(
                text="Set backup account",
                halign="left",
```

(continues on next page)

(continued from previous page)

```
            ),
            MDDialogContentContainer(
                MDListItem(
                    MDListItemLeadingAvatar(
                        source=f"{images_path}/logo/kivymd-icon-128.png",
                    ),
                    MDListItemSupportingText(
                        text="user01@gmail.com",
                    ),
                    theme_bg_color="Custom",
                    md_bg_color=self.theme_cls.transparentColor,
                ),
                MDListItem(
                    MDListItemLeadingAvatar(
                        source="data/logo/kivy-icon-128.png",
                    ),
                    MDListItemSupportingText(
                        text="user01@gmail.com",
                    ),
                    theme_bg_color="Custom",
                    md_bg_color=self.theme_cls.transparentColor,
                ),
                orientation="vertical",
            ),
        ).open()


Example().run()
```



API - kivymd.uix.dialog.dialog

class kivymd.uix.dialog.dialog.**MDDialog**(**args*, ***kwargs*)

Dialog class.

For more information, see in the *MDCard* and *MotionDialogBehavior* classes documentation.

Events

***on_pre_open*:**
Fired before the MDDialog is opened. When this event is fired MDDialog is not yet added to window.

***on_open*:**
Fired when the MDDialog is opened.

***on_pre_dismiss*:**
Fired before the MDDialog is closed.

***on_dismiss*:**
Fired when the MDDialog is closed. If the callback returns True, the dismiss will be canceled.

width_offset

Dialog offset from device width.

width_offset is an NumericProperty and defaults to *dp(48)*.

radius

Dialog corners rounding value.

radius is an VariableListProperty and defaults to *[dp(28), dp(28), dp(28), dp(28)]*.

scrim_color

Color for scrim in (r, g, b, a) or string format.

scrim_color is a ColorProperty and defaults to *[0, 0, 0, 0.5]*.

auto_dismiss

This property determines if the dialog is automatically dismissed when the user clicks outside it.

..versionadded:: 2.0.0

auto_dismiss is a BooleanProperty and defaults to True.

update_width(**args*) → None

add_widget(*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)
```

open() → None
Show the dialog.

on_pre_open(**args*) → None
Fired when a dialog pre opened.

on_open(**args*) → None
Fired when a dialog opened.

on_dismiss(**args*) → None
Fired when a dialog dismiss.

on_pre_dismiss(**args*) → None
Fired when a dialog pre-dismiss.

on_touch_down(*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.

dismiss(**args*) → None
Closes the dialog.

class kivymd.uix.dialog.dialog.**MDDialogIcon**(**args*, ***kwargs*)
The class implements an icon.

For more information, see in the *MDIcon* class documentation.

class kivymd.uix.dialog.dialog.**MDDialogHeadlineText**(**args*, ***kwargs*)
The class implements the headline text.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.dialog.dialog.**MDDialogSupportingText**(**args*, ***kwargs*)
The class implements the supporting text.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.dialog.dialog.**MDDialogContentContainer**(**args*, ***kwargs*)

The class implements the container for custom widgets.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

class kivymd.uix.dialog.dialog.**MDDialogButtonContainer**(**args*, ***kwargs*)

The class implements a container for placing dialog buttons.

For more information, see in the *DeclarativeBehavior* and BoxLayout classes documentation.

2.3.34 SelectionControls

See also:

Material Design spec, Checkbox

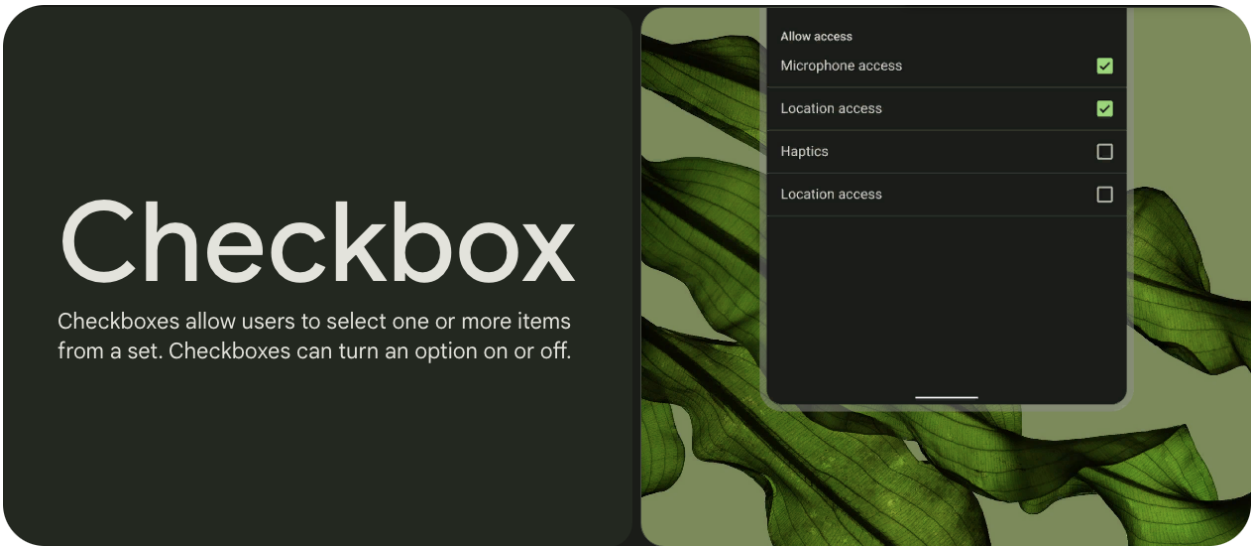
Material Design spec, Switch

Selection controls allow the user to select options.

KivyMD provides the following selection controls classes for use:

- *MDCheckbox*
- *MDSwitch*

MDCheckbox



Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp


KV = '''
MDFloatLayout:

    MDCheckbox:
        size_hint: None, None
        size: "48dp", "48dp"
        pos_hint: {'center_x': .5, 'center_y': .5}
'''


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


Example().run()
```

Note: Be sure to specify the size of the checkbox. By default, it is *(dp(48), dp(48))*, but the ripple effect takes up all the available space.

Control state

```
MDCheckbox:
    on_active: app.on_checkbox_active(*args)
```

```
def on_checkbox_active(self, checkbox, value):
    if value:
        print('The checkbox', checkbox, 'is active', 'and', checkbox.state, 'state')
    else:
        print('The checkbox', checkbox, 'is inactive', 'and', checkbox.state, 'state')
```

MDCheckbox with group

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<Check@MDCheckbox>:
    group: 'group'
    size_hint: None, None
    size: dp(48), dp(48)


MDFloatLayout:

    Check:
        active: True
        pos_hint: {'center_x': .4, 'center_y': .5}

    Check:
        pos_hint: {'center_x': .6, 'center_y': .5}
'''


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


Example().run()
```

Parent and child checkboxes

Checkboxes can have a parent-child relationship with other checkboxes. When the parent checkbox is checked, all child checkboxes are checked. If a parent checkbox is unchecked, all child checkboxes are unchecked. If some, but not all, child checkboxes are checked, the parent checkbox becomes an indeterminate checkbox.

Usage

```
MDCheckbox:
    group: "root"  # this is a required name for the parent checkbox group

MDCheckbox:
    group: "child"  # this is a required name for a group of child checkboxes

MDCheckbox:
    group: "child"  # this is a required name for a group of child checkboxes
```

Example

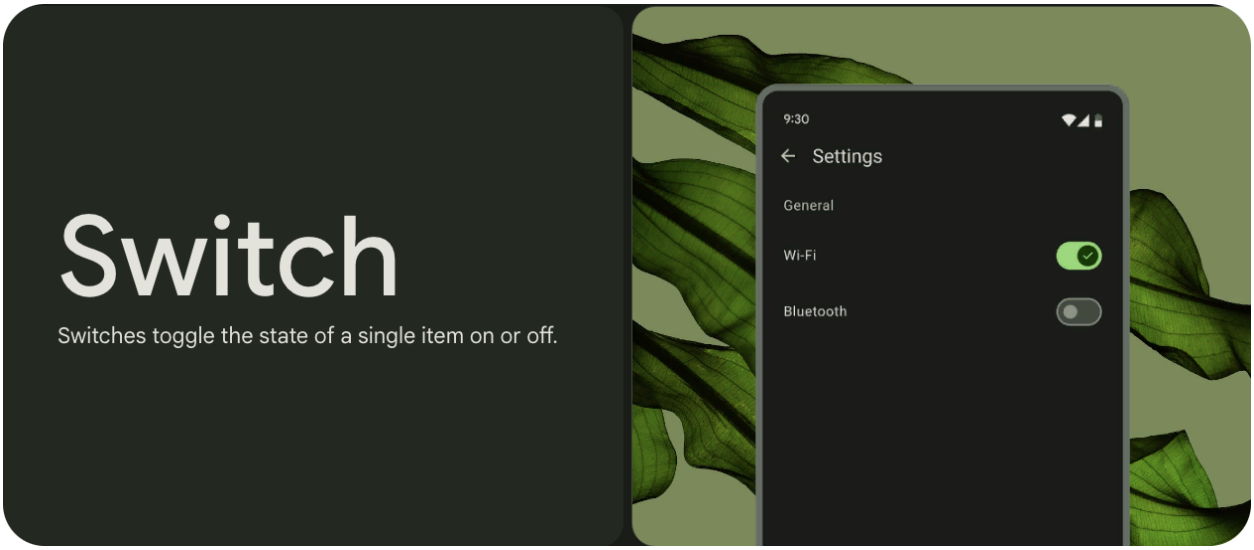
```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout

KV = '''
<CheckItem>
    adaptive_height: True

    MDCheckbox:
        group: root.group

    MDLabel:
        text: root.text
        adaptive_height: True
        padding_x: "12dp"
        pos_hint: {"center_y": .5}


MDBoxLayout:
    orientation: "vertical"
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        orientation: "vertical"
        adaptive_height: True
        padding: "12dp", "36dp", 0, 0
        spacing: "12dp"

        CheckItem:
            text: "Recieve emails"
            group: "root"

        MDBoxLayout:
            orientation: "vertical"
            adaptive_height: True
            padding: "24dp", 0, 0, 0
            spacing: "12dp"

            CheckItem:
                text: "Daily"
                group: "child"

            CheckItem:
                text: "Weekly"
                group: "child"

            CheckItem:
                text: "Monthly"
```

(continues on next page)

(continued from previous page)

```
                group: "child"

    MDWidget:
'''


class CheckItem(MDBoxLayout):
    text = StringProperty()
    group = StringProperty()


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


Example().run()
```

MDSwitch



Usage

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDFloatLayout:

    MDSwitch:
        pos_hint: {'center_x': .5, 'center_y': .5}
'''


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


Example().run()
```

Note: Control state of *MDSwitch* same way as in *MDCheckbox*.

API - kivymd.uix.selectioncontrol.selectioncontrol

class kivymd.uix.selectioncontrol.selectioncontrol.**MDCheckbox**(***kwargs*)

Checkbox class.

For more information, see in the *StateLayerBehavior* and *CircularRippleBehavior* and *ScaleBehavior* and ToggleButtonBehavior and *MDIcon* classes documentation.

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.

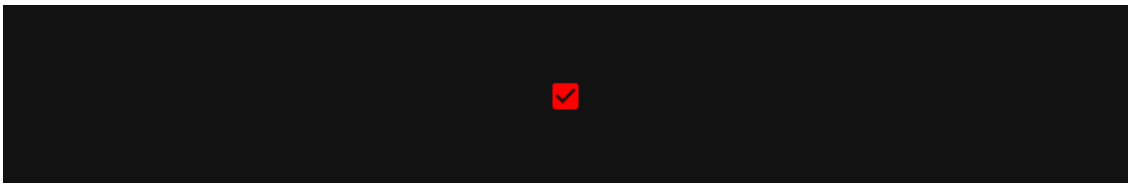
radio_icon_down is a StringProperty and defaults to *'checkbox-marked-circle'*.

color_active

Color in (r, g, b, a) or string format when the checkbox is in the active state.

New in version 1.0.0.

```
MDCheckbox:
    color_active: "red"
```



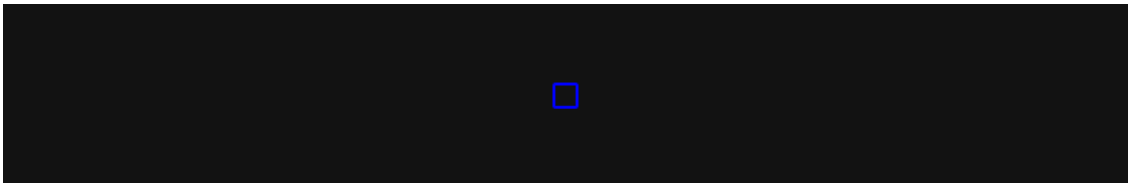
color_active is a ColorProperty and defaults to *None*.

color_inactive

Color in (r, g, b, a) or string format 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*.

color_disabled

Color in (r, g, b, a) or string format when the checkbox is in the disabled state.

New in version 2.0.0: Use *color_disabled* instead.

color_disabled is a ColorProperty and defaults to *None*.

disabled_color

Color in (r, g, b, a) or string format when the checkbox is in the disabled state.

Deprecated since version 2.0.0: Use *color_disabled* instead.

disabled_color is a ColorProperty and defaults to *None*.

selected_color

Color in (r, g, b, a) or string format 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 in (r, g, b, a) or string format 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_icon(**args*) → None

Fired when the values of *checkbox_icon_normal* and *checkbox_icon_down* and *radio_icon_normal* and group change.

set_root_active() → None

set_child_active(*active: bool*)

on_state(**args*) → None

Fired when the values of state change.

on_active(**args*) → None

Fired when the values of *active* change.

on_touch_down(*touch*)

Receive a touch down event.

Parameters

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

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

class kivymd.uix.selectioncontrol.selectioncontrol.**MDSwitch**(***kwargs*)

Switch class.

For more information, see in the *StateLayerBehavior* and *MDFloatLayout* classes documentation.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the switch when the switch is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

ripple_effect

Allows or does not allow the ripple effect when activating/deactivating the switch.

New in version 2.0.0.

ripple_effect is a BooleanProperty and defaults to *True*.

active

Indicates if the switch is active or inactive.

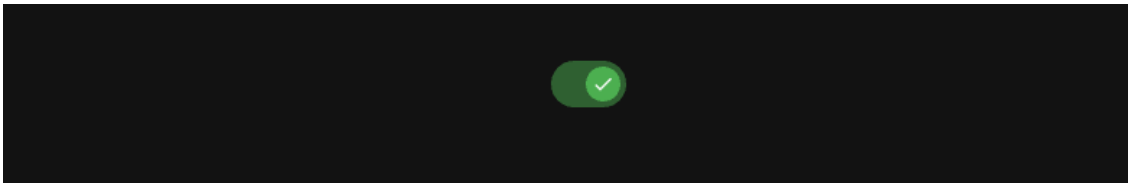
active is a BooleanProperty and defaults to *False*.

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"
```

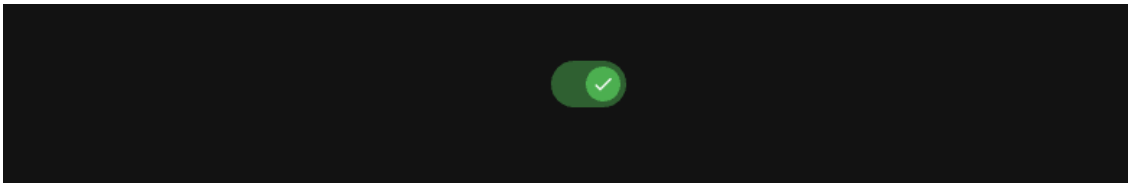
icon_inactive is a StringProperty and defaults to ‘’.

icon_active_color

Thumb icon color in (r, g, b, a) or string format 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"
```



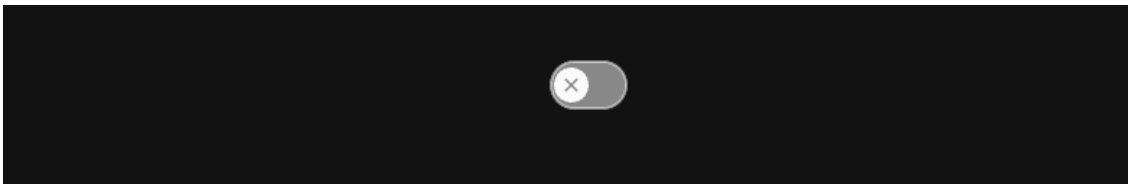
icon_active_color is a ColorProperty and defaults to *None*.

icon_inactive_color

Thumb icon color in (r, g, b, a) or string format 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 in (r, g, b, a) or string format of the thumb when the switch is active.

New in version 1.0.0.

```
MDSwitch:
    active: True
    thumb_color_active: "brown"
```

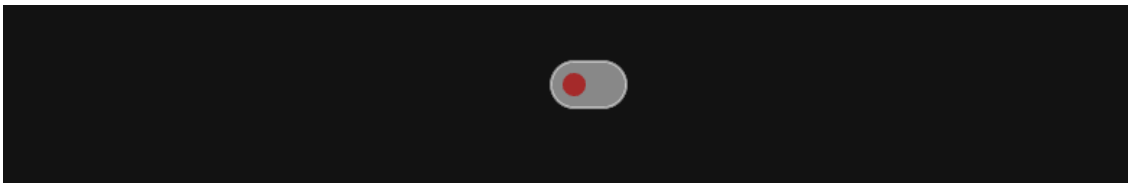
thumb_color_active is an ColorProperty and default to *None*.

thumb_color_inactive

The color in (r, g, b, a) or string format 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 in (r, g, b, a) or string format of the thumb when the switch is in the disabled state.

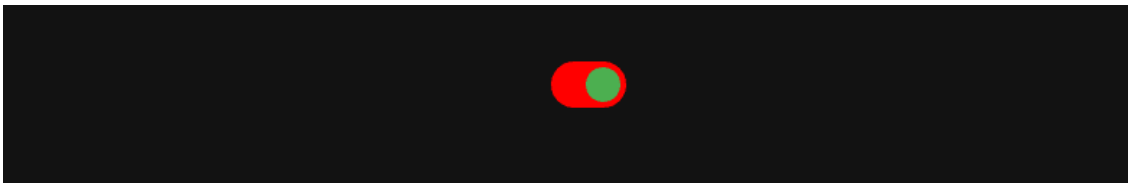
```
MDSwitch:
    active: True
    thumb_color_disabled: "brown"
    disabled: True
```

thumb_color_disabled is an ColorProperty and default to *None*.

track_color_active

The color in (r, g, b, a) or string format 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 in (r, g, b, a) or string format of the track when the switch is inactive.

New in version 1.0.0.

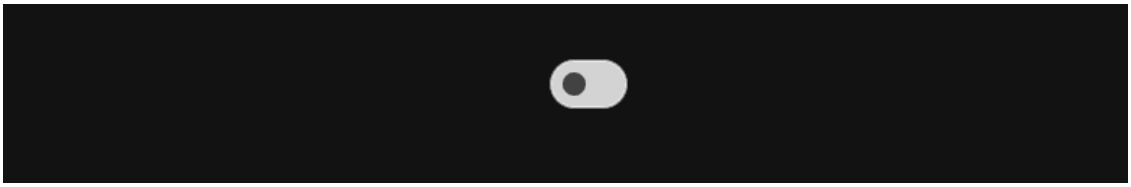
```
MDSwitch:
    track_color_inactive: "red"
```

track_color_inactive is an ColorProperty and default to *None*.

track_color_disabled

The color in (r, g, b, a) or string format 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*.

line_color_disabled

The color of the outline in the disabled state

New in version 2.0.0.

line_color_disabled is an ColorProperty and defaults to *None*.

set_icon(*instance_switch*, *icon_value: str*) → None

Fired when the values of *icon_active* and *icon_inactive* change.

on_line_color(*instance*, *value*) → None

Fired when the values of line_color change.

on_active(*instance_switch*, *active_value: bool*) → None

Fired when the values of *active* change.

on_thumb_down() → None

Fired 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.

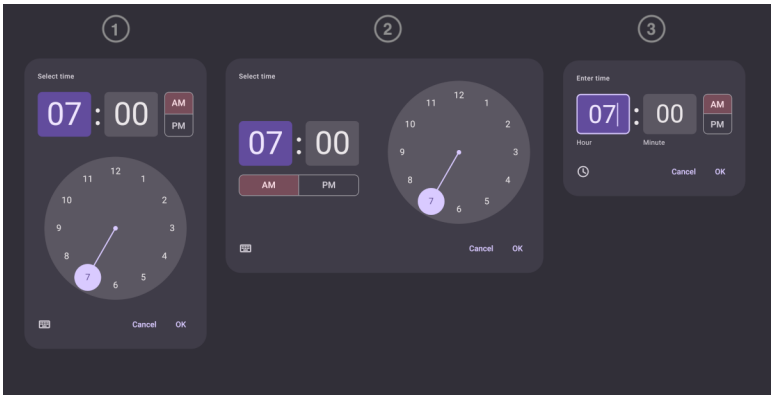
2.3.35 TimePicker

See also:

Material Design spec, Date picker

Time pickers help users select and set a specific time.

- Time pickers are modal and cover the main content
- Two types: dial and input
- Users can select hours, minutes, or periods of time
- Make sure time can easily be selected by hand on a mobile device



1. Vertical dial time picker
2. Horizontal dial time picker
3. Time picker input

KivyMD provides the following date pickers classes for use:

- *MDTimePickerDialVertical*
- *MDTimePickerDialHorizontal*
- *MDTimePickerInput*

MDTimePickerDialVertical

Time pickers allow people to enter a specific time value. They're displayed in dialogs and can be used to select hours, minutes, or periods of time.

They can be used for a wide range of scenarios. Common use cases include:

- Setting an alarm
- Scheduling a meeting

Time pickers are not ideal for nuanced or granular time selection, such as milliseconds for a stopwatch application.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.pickers import MDTimePickerDialVertical

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_time_picker()

        MDButtonText:
            text: "Open time picker"
'''
```

(continues on next page)

(continued from previous page)

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

    def show_time_picker(self):
        time_picker = MDTimePickerDialVertical()
        time_picker.open()


Example().run()
```

MDTimePickerDialHorizontal

The clock dial interface adapts to a device's orientation. In landscape mode, the stacked input and selection options are positioned side-by-side.

```
def show_time_picker(self):
    MDTimePickerDialHorizontal().open()
```

Note: You must control the orientation of the time picker yourself.

```
from typing import Literal

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

from kivymd.app import MDApp
from kivymd.theming import ThemeManager
from kivymd.uix.pickers import (
    MDTimePickerDialHorizontal,
    MDTimePickerDialVertical,
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release:
            app.open_time_picker_horizontal("1", "10")                if self.theme_cls.
↪device_orientation == "landscape" else                 app.open_time_picker_vertical("1
↪", "10")
```

(continues on next page)

(continued from previous page)

```
        MDButtonText:
            text: "Open time picker"
'''


class Example(MDApp):
    ORIENTATION = Literal["portrait", "landscape"]
    time_picker_horizontal: MDTimePickerDialHorizontal = ObjectProperty(
        allownone=True
    )
    time_picker_vertical: MDTimePickerDialHorizontal = ObjectProperty(
        allownone=True
    )

    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.bind(device_orientation=self.check_orientation)
        return Builder.load_string(KV)

    def check_orientation(
        self, instance: ThemeManager, orientation: ORIENTATION
    ):
        if orientation == "portrait" and self.time_picker_horizontal:
            self.time_picker_horizontal.dismiss()
            hour = str(self.time_picker_horizontal.time.hour)
            minute = str(self.time_picker_horizontal.time.minute)
            Clock.schedule_once(
                lambda x: self.open_time_picker_vertical(hour, minute),
                0.1,
            )
        elif orientation == "landscape" and self.time_picker_vertical:
            self.time_picker_vertical.dismiss()
            hour = str(self.time_picker_vertical.time.hour)
            minute = str(self.time_picker_vertical.time.minute)
            Clock.schedule_once(
                lambda x: self.open_time_picker_horizontal(hour, minute),
                0.1,
            )

    def open_time_picker_horizontal(self, hour, minute):
        self.time_picker_vertical = None
        self.time_picker_horizontal = MDTimePickerDialHorizontal(
            hour=hour, minute=minute
        )
        self.time_picker_horizontal.open()

    def open_time_picker_vertical(self, hour, minute):
        self.time_picker_horizontal = None
        self.time_picker_vertical = MDTimePickerDialVertical(
            hour=hour, minute=minute
        )
```

(continues on next page)

(continued from previous page)

```
        self.time_picker_vertical.open()


Example().run()
```

MDTimePickerInput

Time input pickers allow people to specify a time using keyboard numbers. This input option should be accessible from any other mobile time picker interface by tapping the keyboard icon.

```
def show_time_picker(self):
    MDTimePickerInput().open()
```

Events

on_edit event

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

from kivymd.app import MDApp
from kivymd.uix.pickers import MDTimePickerDialVertical, MDTimePickerInput

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_time_picker_vertical()

        MDButtonText:
            text: "Open time picker"
'''


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

    def on_edit_time_picker_input(self, time_picker_input):
        time_picker_input.dismiss()
        Clock.schedule_once(self.show_time_picker_vertical, 0.2)

    def show_time_picker_input(self, *args):
```

(continues on next page)

(continued from previous page)

```
        time_picker_input = MDTimePickerInput()
        time_picker_input.bind(on_edit=self.on_edit_time_picker_input)
        time_picker_input.open()

    def on_edit_time_picker_vertical(self, time_picker_vertical):
        time_picker_vertical.dismiss()
        Clock.schedule_once(self.show_time_picker_input, 0.2)

    def show_time_picker_vertical(self, *args):
        time_picker_vertical = MDTimePickerDialVertical()
        time_picker_vertical.bind(on_edit=self.on_edit_time_picker_vertical)
        time_picker_vertical.open()


Example().run()
```

on_hour_select event

```
def on_hour_select(
    self, time_picker_vertical: MDTimePickerDialVertical, mode: str
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"On '{mode}' select",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_hour_select=self.on_hour_select)
    time_picker_vertical.open()
```

on_minute_select event

```
def on_minute_select(
    self, time_picker_vertical: MDTimePickerDialVertical, mode: str
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"On '{mode}' select",
        ),
```

(continues on next page)

(continued from previous page)

```
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_minute_select=self.on_minute_select)
    time_picker_vertical.open()
```

on_am_pm event

```
def on_am_pm(
    self, time_picker_vertical: MDTimePickerDialVertical, am_pm: str
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"'{am_pm.upper()}' select",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_am_pm=self.on_am_pm)
    time_picker_vertical.open()
```

on_selector_hour event

```
def on_selector_hour(
    self, time_picker_vertical: MDTimePickerDialVertical, hour: str
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"The value of the hour is `{hour}` select",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()
```

(continues on next page)

(continued from previous page)

```
def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_selector_hour=self.on_selector_hour)
    time_picker_vertical.open()
```

on_selector_minute event

```
def on_selector_minute(
    self, time_picker_vertical: MDTimePickerDialVertical, minute: str
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"The value of the hour is `{minute}` select",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_selector_minute=self.on_selector_minute)
    time_picker_vertical.open()
```

on_cancel event

```
def on_cancel(
    self, time_picker_vertical: MDTimePickerDialVertical
):
    time_picker_vertical.dismiss()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_cancel=self.on_cancel)
    time_picker_vertical.open()
```

on_ok event

```
def on_ok(
    self, time_picker_vertical: MDTimePickerDialVertical
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"Time is `{time_picker_vertical.time}`",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerDialVertical()
    time_picker_vertical.bind(on_ok=self.on_ok)
    time_picker_vertical.open()
```

on_time_input event

```
def on_time_input(
    self,
    time_picker_vertical: MDTimePickerInput,
    type_time: str,
    value: str,
):
    MDSnackbar(
        MDSnackbarSupportingText(
            text=f"The {type_time} value is set to {value}",
        ),
        y=dp(24),
        orientation="horizontal",
        pos_hint={"center_x": 0.5},
        size_hint_x=0.5,
    ).open()

def show_time_picker_vertical(self, *args):
    time_picker_vertical = MDTimePickerInput()
    time_picker_vertical.bind(on_time_input=self.on_time_input)
    time_picker_vertical.open()
```

API break

1.2.0 version

```
time_picker_dialog = MDTimePicker()
time_picker_dialog.open()
```

2.0.0 version

```
# time_picker_dialog = MDTimePickerDialVertical()
# time_picker_dialog = MDTimePickerDialHorizontal()

time_picker_dialog = MDTimePickerInput()
time_picker_dialog.open()
```

API - kivymd.uix.pickers.timepicker.timepicker

class kivymd.uix.pickers.timepicker.timepicker.**MDBaseTimePicker**(***kwargs*)

Implements the base class of the time picker.

New in version 2.0.0.

For more information, see in the *ThemableBehavior* and *MotionTimePickerBehavior* and BoxLayout and classes documentation.

Events

on_cancel
Fired when the 'Cancel' button is pressed.

on_ok
Fired when the 'Ok' button is pressed.

on_dismiss
Fired when a date picker closes.

on_edit
Fired when you click on the date editing icon.

on_hour_select
Fired when the hour input field container is clicked.

on_minute_select
Fired when the minute input field container is clicked.

on_am_pm
Fired when the AP/PM switching elements are pressed.

on_selector_hour
Fired when switching the hour value in the clock face container.

on_selector_minute
Fired when switching the minute value in the clock face container.

hour

Current hour.

hour is an StringProperty and defaults to *'12'*.

minute

Current minute.

minute is an StringProperty and defaults to *0*.

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*.

headline_text

Headline text.

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

text_button_ok

The text of the confirmation button.

text_button_ok is a StringProperty and defaults to *'Ok'*.

text_button_cancel

The text of the cancel button.

text_button_cancel is a StringProperty and defaults to *'Cancel'*.

radius

Container radius.

radius is an VariableListProperty and defaults to *[dp(16), dp(16), dp(16), dp(16)]*.

is_open

Is the date picker dialog open.

is_open is a BooleanProperty and defaults to *False*.

scrim_color

Color for scrim in (r, g, b, a) or string format.

scrim_color is a ColorProperty and defaults to *[0, 0, 0, 0.5]*.

set_time(*time_obj: datetime.time*) → None

Manually set time dialog with the specified time.

open() → None

Show the dialog time picker.

on_touch_down(*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.

dismiss(**args*) → None

Dismiss the dialog time picker.

on_dismiss(**args*) → None

Fired when a time picker closes.

on_cancel(**args*) → None

Fired when the 'Cancel' button is pressed.

on_ok(**args*) → None

Fired when the 'Ok' button is pressed.

on_hour_select(**args*) → None

Fired when the hour input field container is clicked.

on_minute_select(**args*) → None

Fired when the minute input field container is clicked.

on_am_pm(**args*) → None

Fired when the AP/PM switching elements are pressed.

on_edit(**args*) → None

Fired when you click on the time editing icon.

on_selector_hour(**args*) → None

Fired when switching the hour value in the clock face container.

on_selector_minute(**args*) → None

Fired when switching the minute value in the clock face container.

on_time_input(**args*) → None

Fired when switching the minute value in the clock face container.

class kivymd.uix.pickers.timepicker.timepicker.**MDTimePickerInput**(***kwargs*)

Implements input time picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseTimePicker* classes documentation.

class kivymd.uix.pickers.timepicker.timepicker.**MDTimePickerDialVertical**(***kwargs*)

Implements vertical time picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseTimePicker* classes documentation.

class kivymd.uix.pickers.timepicker.timepicker.**MDTimePickerDialHorizontal**(***kwargs*)

Implements horizontal time picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseTimePicker* classes documentation.

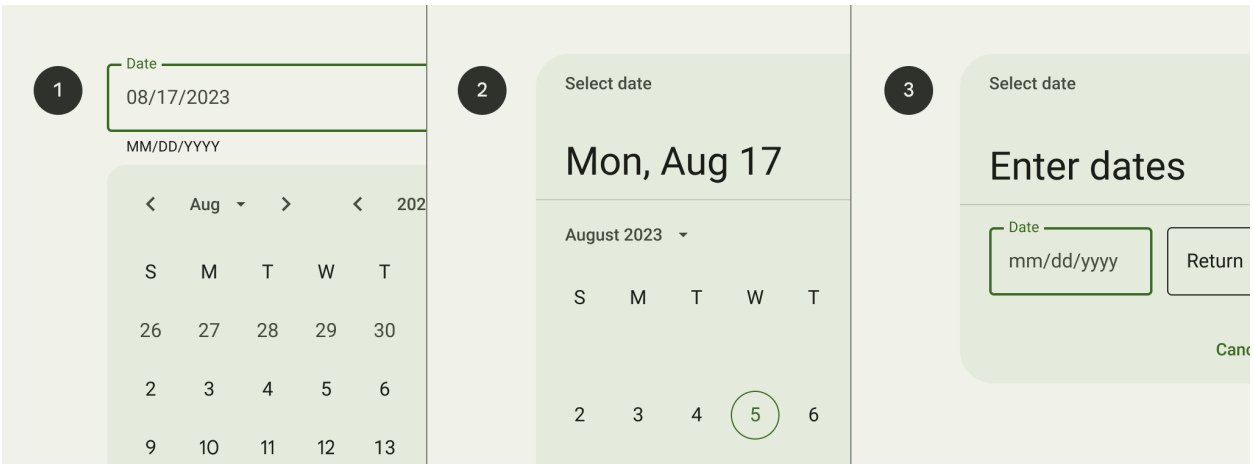
2.3.36 DatePicker

See also:

Material Design spec, Date picker

Date pickers let people select a date, or a range of dates.

- Date pickers can display past, present, or future dates
- Three types: docked, modal, modal input
- Clearly indicate important dates, such as current and selected days
- Follow common patterns, like a calendar view



1. Docked date picker
2. Modal date picker

3. Modal date input

KivyMD provides the following date pickers classes for use:

- *MDDockedDatePicker*
- *MDModalDatePicker*
- *MDModalInputDatePicker*

MDDockedDatePicker

Docked datepickers allow the selection of a specific date and year. The docked datepicker displays a date input field by default, and a dropdown calendar appears when the user taps on the input field. Either form of date entry can be interacted with.

Docked date pickers are ideal for navigating dates in both the near future or past and the distant future or past, as they provide multiple ways to select dates.

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

from kivymd.app import MDApp
from kivymd.uix.pickers import MDDockedDatePicker

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTextField:
        id: field
        mode: "outlined"
        pos_hint: {'center_x': .5, 'center_y': .85}
        size_hint_x: .5
        on_focus: app.show_date_picker(self.focus)

        MDTextFieldHintText:
            text: "Docked date picker"

        MDTextFieldHelperText:
            text: "MM/DD/YYYY"
            mode: "persistent"

        MDTextFieldTrailingIcon:
            icon: "calendar"
'''


class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)
```

(continues on next page)

(continued from previous page)

```
    def show_date_picker(self, focus):
        if not focus:
            return

        date_dialog = MDDockedDatePicker()
        # You have to control the position of the date picker dialog yourself.
        date_dialog.pos = [
            self.root.ids.field.center_x - date_dialog.width / 2,
            self.root.ids.field.y - (date_dialog.height + dp(32)),
        ]
        date_dialog.open()


Example().run()
```

MDModalDatePicker

Modal date pickers navigate across dates in several ways:

- To navigate across months, swipe horizontally (not implemented in KivyMD)
- To navigate across years, scroll vertically (not implemented in KivyMD)
- To access the year picker, tap the year

Don't use a modal date picker to prompt for dates in the distant past or future, such as a date of birth. In these cases, use a modal input picker or a docked datepicker instead.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.pickers import MDModalDatePicker

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_date_picker()

        MDButtonText:
            text: "Open modal date picker dialog"
'''


class Example(MDApp):
    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)
```

(continues on next page)

(continued from previous page)

```
    def show_date_picker(self):
        date_dialog = MDModalDatePicker()
        date_dialog.open()


Example().run()
```

MDModalInputDatePicker

Modal date inputs allow the manual entry of dates using the numbers on a keyboard. Users can input a date or a range of dates in a dialog.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.pickers import MDModalInputDatePicker

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_date_picker()

        MDButtonText:
            text: "Open modal date picker dialog"
'''


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

    def show_date_picker(self):
        date_dialog = MDModalInputDatePicker()
        date_dialog.open()


Example().run()
```

The range of available dates

To display only the selected date range, use the *min_date* and *max_date* parameters:

```
def show_modal_date_picker(self, *args):
    MDModalDatePicker(
        mark_today=False,
        min_date=datetime.date.today(),
        max_date=datetime.date(
            datetime.date.today().year,
            datetime.date.today().month,
            datetime.date.today().day + 4,
        ),
    ).open()
```

Only dates in the specified range will be available for selection:

Select the date range

To select the date range, use the *mode* parameter with the value “range”:

```
def show_modal_date_picker(self, *args):
    MDModalDatePicker(mode="range").open()
```

Setting the date range manually

```
def show_modal_date_picker(self, *args):
    MDModalInputDatePicker(mode="range").open()
```

Events

on_edit event

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

from kivymd.app import MDApp
from kivymd.uix.pickers import MDModalInputDatePicker, MDModalDatePicker

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

```

(continues on next page)

(continued from previous page)

```
    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_modal_date_picker()

        MDButtonText:
            text: "Open modal date picker dialog"
'''


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

    def show_modal_input_date_picker(self, *args):
        def on_edit(*args):
            date_dialog.dismiss()
            Clock.schedule_once(self.show_modal_date_picker, 0.2)

        date_dialog = MDModalInputDatePicker()
        date_dialog.bind(on_edit=on_edit)
        date_dialog.open()

    def on_edit(self, instance_date_picker):
        instance_date_picker.dismiss()
        Clock.schedule_once(self.show_modal_input_date_picker, 0.2)

    def show_modal_date_picker(self, *args):
        date_dialog = MDModalDatePicker()
        date_dialog.bind(on_edit=self.on_edit)
        date_dialog.open()


Example().run()
```

on_select_day event

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

from kivymd.app import MDApp
from kivymd.uix.pickers import MDModalDatePicker
from kivymd.uix.snackbar import MDSnackbar, MDSnackbarSupportingText

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
```

(continues on next page)

(continued from previous page)

```
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_modal_date_picker()

        MDButtonText:
            text: "Open modal date picker dialog"
'''


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

    def on_select_day(self, instance_date_picker, number_day):
        instance_date_picker.dismiss()
        MDSnackbar(
            MDSnackbarSupportingText(
                text=f"The selected day is {number_day}",
            ),
            y=dp(24),
            orientation="horizontal",
            pos_hint={"center_x": 0.5},
            size_hint_x=0.5,
            background_color="olive"
        ).open()

    def show_modal_date_picker(self, *args):
        date_dialog = MDModalDatePicker()
        date_dialog.bind(on_select_day=self.on_select_day)
        date_dialog.open()


Example().run()
```

on_select_month event

```
def on_select_month(self, instance_date_picker, number_month):
    [...]

def show_modal_date_picker(self, *args):
    [...]
    date_dialog.bind(on_select_month=self.on_select_month)
    [...]
```

on_select_year event

```
def on_select_year(self, instance_date_picker, number_year):
    [...]

def show_modal_date_picker(self, *args):
    [...]
    date_dialog.bind(on_select_month=self.on_select_year)
    [...]
```

on_cancel event

```
def on_cancel(self, instance_date_picker):
    [...]

def show_modal_date_picker(self, *args):
    [...]
    date_dialog.bind(on_cancel=self.on_cancel)
    [...]
```

on_ok event

```
def on_ok(self, instance_date_picker):
    print(instance_date_picker.get_date()[0])

def show_modal_date_picker(self, *args):
    [...]
    date_dialog.bind(on_ok=self.on_ok)
    [...]
```

on_ok with range event

```
import datetime

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

from kivymd.app import MDApp
from kivymd.uix.pickers import MDModalDatePicker
from kivymd.uix.snackbar import (
    MDSnackbar, MDSnackbarSupportingText, MDSnackbarText
```

(continues on next page)

(continued from previous page)

```
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDButton:
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: app.show_modal_date_picker()

        MDButtonText:
            text: "Open modal date picker dialog"
'''


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

    def on_ok(self, instance_date_picker):
        MDSnackbar(
            MDSnackbarText(
                text="Selected dates is:",
            ),
            MDSnackbarSupportingText(
                text="\n".join(str(date) for date in instance_date_picker.get_date()),
                padding=[0, 0, 0, dp(12)],
            ),
            y=dp(124),
            pos_hint={"center_x": 0.5},
            size_hint_x=0.5,
            padding=[0, 0, "8dp", "8dp"],
        ).open()

    def show_modal_date_picker(self, *args):
        date_dialog = MDModalDatePicker(
            mode="range",
            min_date=datetime.date.today(),
            max_date=datetime.date(
                datetime.date.today().year,
                datetime.date.today().month,
                datetime.date.today().day + 4,
            ),
        )
        date_dialog.bind(on_ok=self.on_ok)
        date_dialog.open()


Example().run()
```

API break

1.2.0 version

```
date_dialog = MDDatePicker()
date_dialog.open()
```

2.0.0 version

```
# date_dialog = MDModalDatePicker()
# date_dialog = MDModalInputDatePicker()

date_dialog = MDDockedDatePicker()
date_dialog.open()
```

API - kivymd.uix.pickers.datepicker.datepicker

class kivymd.uix.pickers.datepicker.datepicker.**MDBaseDatePicker**(*year=None*, *month=None*, *day=None*, *firstweekday=0*, ***kwargs*)

Implements the base class of the date picker.

New in version 2.0.0.

For more information, see in the *ThemableBehavior* and *MotionDatePickerBehavior* and BoxLayout and classes documentation.

Events

on_select_day
Fired when a day is selected.

on_select_month
Fired when a month is selected.

on_select_year
Fired when a year is selected.

on_cancel
Fired when the 'Cancel' button is pressed.

on_ok
Fired when the 'Ok' button is pressed.

on_edit
Fired when you click on the date editing icon.

on_dismiss
Fired when a date picker closes.

day

The day of the month to be opened by default. If not specified, the current number will be used.

day is an NumericProperty and defaults to *0*.

month

The number of month to be opened by default. If not specified, the current number will be used.

month is an NumericProperty and defaults to *0*.

year

The year of month to be opened by default. If not specified, the current number will be used.

year is an NumericProperty and defaults to *0*.

min_year

The year of month to be opened by default. If not specified, the current number will be used.

min_year is an NumericProperty and defaults to *1914*.

max_year

The year of month to be opened by default. If not specified, the current number will be used.

max_year is an NumericProperty and defaults to *2121*.

mode

Dialog type. Available options are: *'picker'*, *'range'*.

mode is an OptionProperty and defaults to *picker*.

min_date

The minimum value of the date range for the *'mode'* parameter. Must be an object <class 'datetime.date'>.

min_date is an ObjectProperty and defaults to *None*.

max_date

The minimum value of the date range for the *'mode'* parameter. Must be an object <class 'datetime.date'>.

max_date is an ObjectProperty and defaults to *None*.

radius

Container radius.

radius is an VariableListProperty and defaults to *[dp(16), dp(16), dp(16), dp(16)]*.

scrim_color

Color for scrim in (r, g, b, a) or string format.

scrim_color is a ColorProperty and defaults to *[0, 0, 0, 0.5]*.

supporting_text

Supporting text.

supporting_text is a StringProperty and defaults to *'Select date'*.

text_button_ok

The text of the confirmation button.

text_button_ok is a StringProperty and defaults to *'Ok'*.

text_button_cancel

The text of the cancel button.

text_button_cancel is a StringProperty and defaults to *'Cancel'*.

mark_today

Highlights the current day.

mark_today is a BooleanProperty and defaults to *True*.

is_open

Is the date picker dialog open.

is_open is a BooleanProperty and defaults to *False*.

sel_year

sel_month

sel_day

calendar_layout

get_date(**args*) → list

Returns a list of dates in the format [datetime.date(yyyy, mm, dd), ...]. The list has two dates if you use a date interval.

set_text_full_date() → str

Returns a string like "Tue, Feb 2".

compare_date_range() → None

change_month(*operation: str*) → None

Called when "chevron-left" and "chevron-right" buttons are pressed. Switches the calendar to the previous/next month.

generate_list_widgets_days() → None

update_calendar(*year*, *month*) → None

set_selected_widget(*widget*) → None

restore_calendar_layout_properties() → None

set_calendar_layout_properties(*method*) → None

dismiss(**args*) → None

Dismiss the dialog date picker.

open() → None

Show the dialog date picker.

on_touch_down(*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_select_day(**args*) → None

Fired when a day is selected.

on_select_month(**args*) → None

Fired when a month is selected.

on_select_year(**args*) → None

Fired when a year is selected.

on_cancel(**args*) → None

Fired when the 'Cancel' button is pressed.

on_ok(**args*) → None

Fired when the 'Ok' button is pressed.

on_edit(**args*) → None

Fired when you click on the date editing icon.

on_dismiss(**args*) → None

Fired when a date picker closes.

class kivymd.uix.pickers.datepicker.datepicker.**MDDockedDatePicker**(***kwargs*)

Implements docked date picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseDatePicker* classes documentation.

generate_menu_month_year_selection(*menu_type: str = 'month'*) → None

Generates a list for the month or year selection menu.

open_close_menu_month_year_selection(*state: bool = True*, *menu_type: str = 'month'*) → None

Hides the calendar layout and opens the list to select the month or year.

class kivymd.uix.pickers.datepicker.datepicker.**MDModalDatePicker**(***kwargs*)

Implements modal date picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseDatePicker* classes documentation.

open() → None

Show the dialog date picker.

generate_list_widgets_years() → None

open_menu_year_selection(**args*) → None

class kivymd.uix.pickers.datepicker.datepicker.**MDModalInputDatePicker**(**args*, ***kwargs*)

Implements modal input date picker.

New in version 2.0.0.

For more information, see in the *CommonElevationBehavior* and *MDBaseDatePicker* classes documentation.

date_format

Format of date strings that will be entered. Available options are: *'dd/mm/yyyy'*, *'mm/dd/yyyy'*, *'yyyy/mm/dd'*.

date_format is an OptionProperty and defaults to *None*.

default_input_date

If true, the current date will be set in the input field.

default_input_date is a BooleanProperty and defaults to *True*.

error_text

Error text when the date entered by the user is not valid.

error_text is a StringProperty and defaults to *'Invalid date format'*.

supporting_input_text

Auxiliary text when entering the date manually.

supporting_input_text is a StringProperty and defaults to *'Enter date'*.

generate_list_widgets_days() → None

update_calendar(**args*) → None

set_input_date(*input_date: str*) → None

get_date(**args*) → list

Returns a list of dates in the format [datetime.date(yyyy, mm, dd), ...]. The list has two dates if you use a date interval.

get_current_date_from_format() → str

Returns the date according to the set format in *date_format*.

open() → None

Show the dialog date picker.

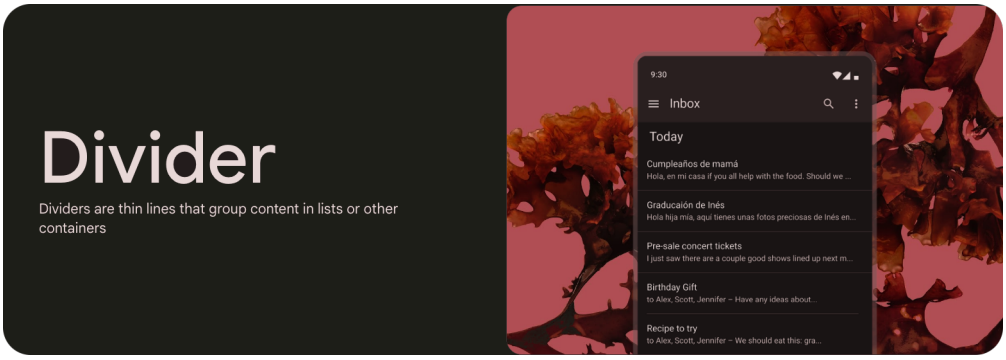
2.3.37 Divider

New in version 2.0.0.

See also:

Material Design 3 spec, Divider

Dividers are thin lines that group content in lists or other containers.



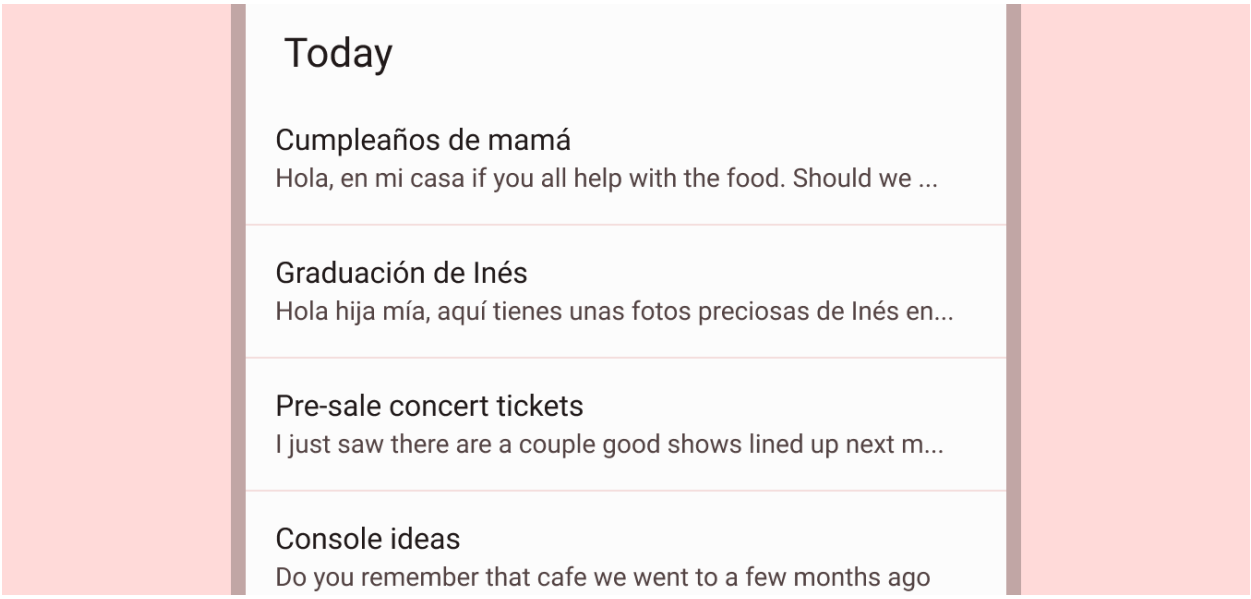
- Make dividers visible but not bold
- Only use dividers if items can't be grouped with open space
- Use dividers to group things, not separate individual items

KivyMD provides the following bar positions for use:

- *HorizontalDivider*
- *VerticalDivider*

HorizontalDivider

Dividers are one way to visually group components and create hierarchy. They can also be used to imply nested parent/child relationships.

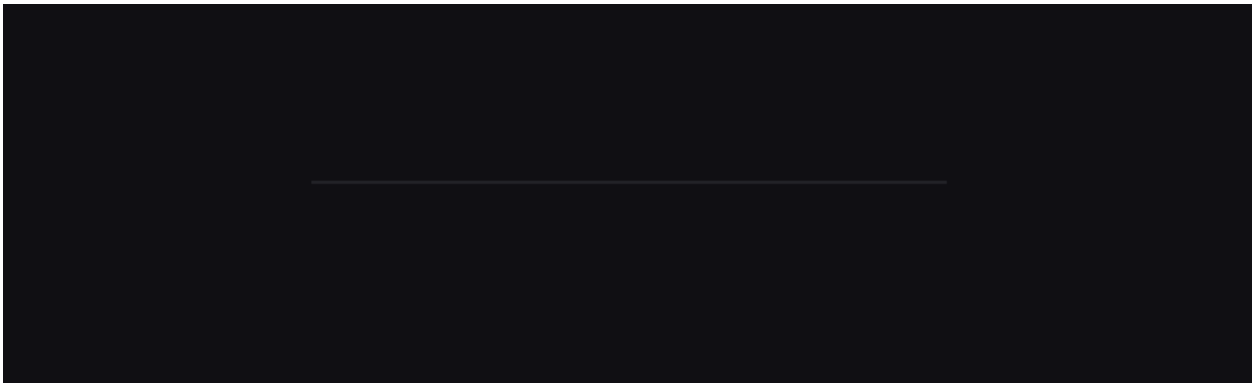



```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
```

(continues on next page)

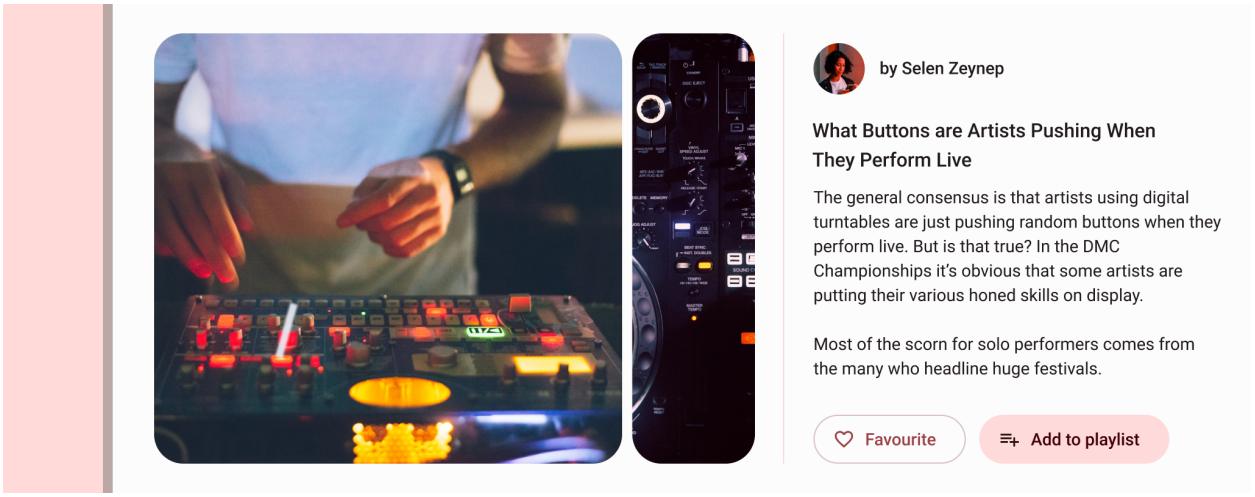
(continued from previous page)

```
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDDivider:
        size_hint_x: .5
        pos_hint: {'center_x': .5, 'center_y': .5}
'''


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


Example().run()
```



VerticalDivider

A vertical divider can be used to arrange content on a larger screen, such as separating paragraph text from video or imagery media.



```
MDDivider:
    size_hint_y: .5
    orientation: "vertical"
```

API break

1.2.0 version

```
MDSeparator:
    [...]
```

2.0.0 version

```
MDDivider:
    [...]
```

API - kivymd.uix.divider.divider

class kivymd.uix.divider.divider.**MDDivider**(***kwargs*)

A divider line.

New in version 2.0.0.

For more information, see in the BoxLayout class documentation.

color

Divider color in (r, g, b, a) or string format.

color is a ColorProperty and defaults to *None*.

divider_width

Divider width.

divider_width is an NumericProperty and defaults to *dp(1)*.

on_orientation(**args*) → None

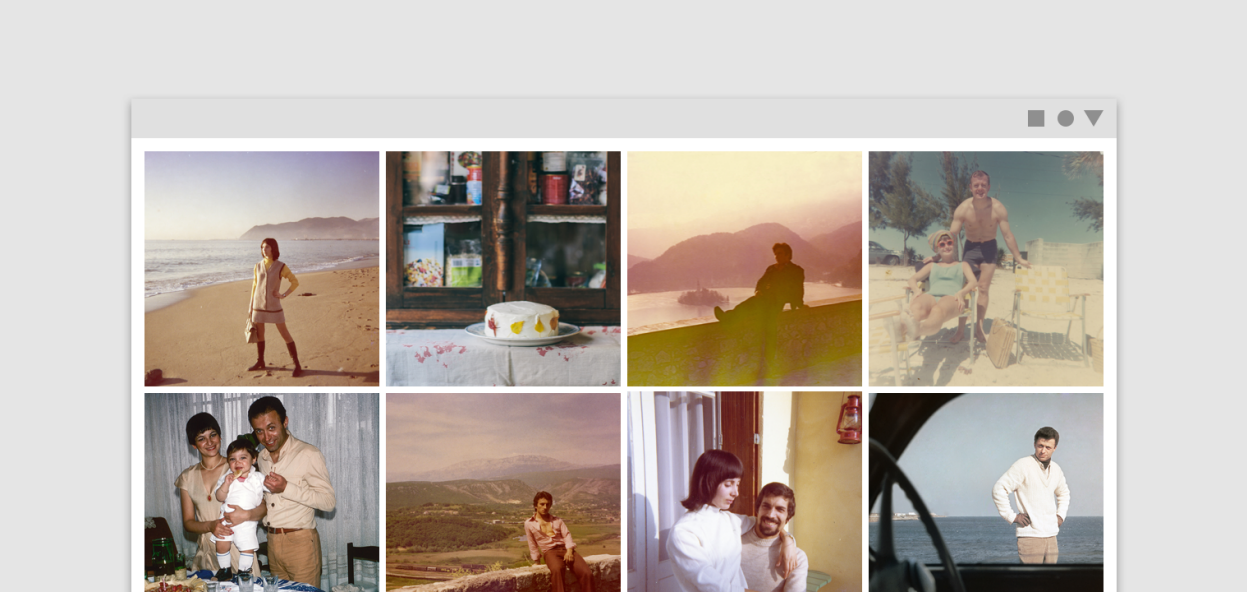
Fired when the values of orientation change.

2.3.38 ImageList

See also:

Material Design spec, Image lists

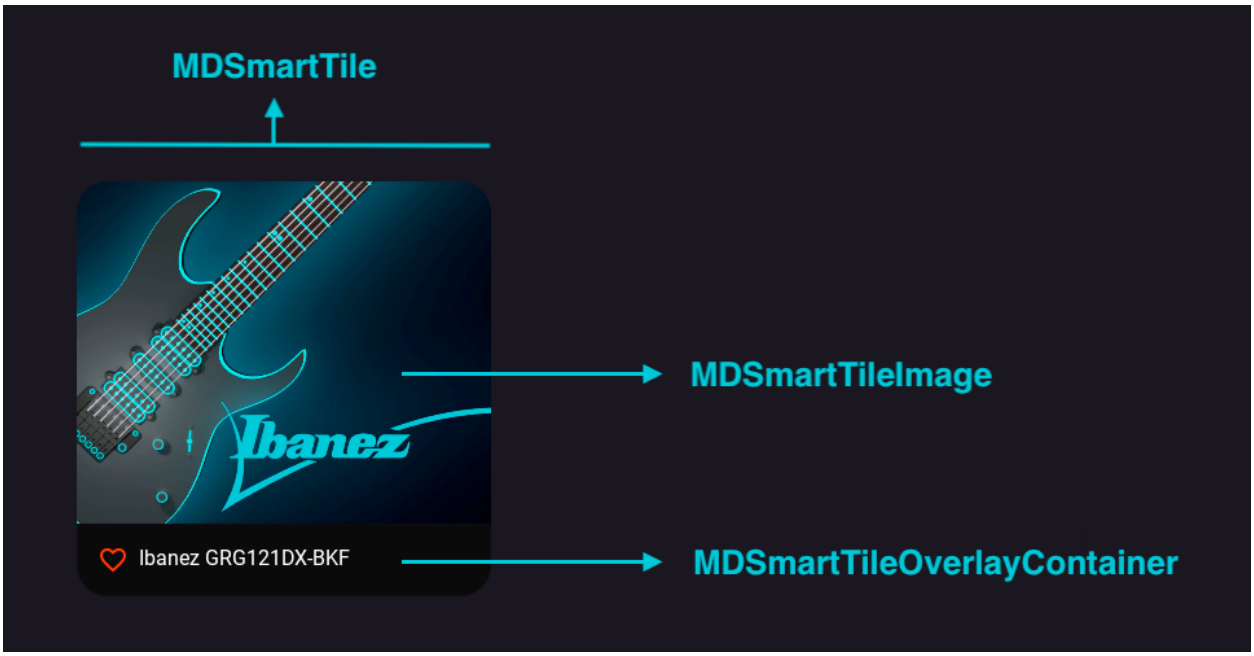
Image lists display a collection of images in an organized grid.



Usage

```
MDSmartTile:
    [...]

    MDSmartTileImage:
        [...]

    MDSmartTileOverlayContainer:
        [...]

        # Content
        [...]
```

Anatomy



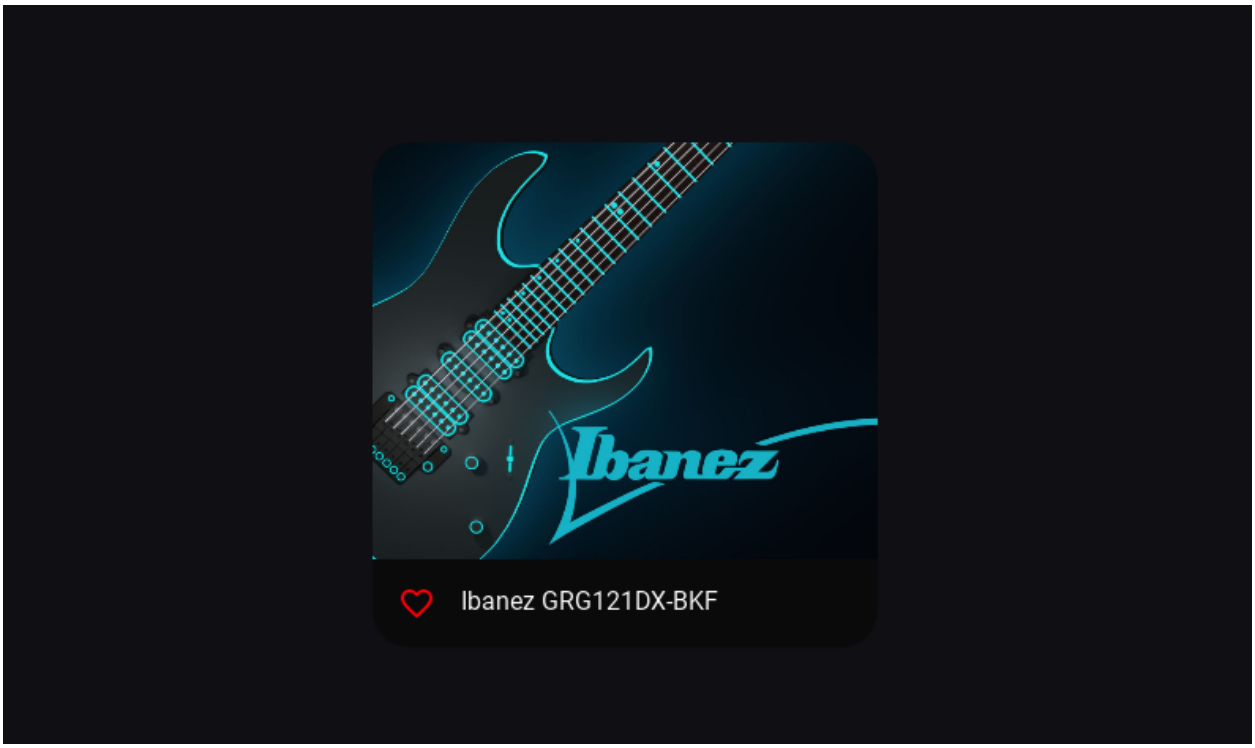
Example

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDSmartTile:
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint: None, None
        size: "320dp", "320dp"
        overlap: False

        MDSmartTileImage:
            source: "bg.jpg"
            radius: [dp(24), dp(24), 0, 0]

        MDSmartTileOverlayContainer:
            md_bg_color: 0, 0, 0, .5
            adaptive_height: True
            padding: "8dp"
            spacing: "8dp"
            radius: [0, 0, dp(24), dp(24)]

            MDIconButton:
```

(continues on next page)

(continued from previous page)

```
                icon: "heart-outline"
                theme_icon_color: "Custom"
                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: "Ibanez GRG121DX-BKF"
                theme_text_color: "Custom"
                text_color: "white"
'''


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


Example().run()
```



API break

1.2.0 version

```
MDSmartTile:
    [...]

    # Content.
    MDIconButton:
        [...]

    MDLabel:
        [...]
```

2.0.0 version

```
MDSmartTile:
    [...]

    MDSmartTileImage:
        [...]

    MDSmartTileOverlayContainer:
        [...]

        # Content.
        [...]
```

API - kivymd.uix.imagelist.imagelist

class kivymd.uix.imagelist.imagelist.**MDSmartTileImage**(***kwargs*)

Implements the tile image.

Changed in version 2.0.0: The *SmartTileImage* class has been renamed to *MDSmartTileImage*.

For more information, see in the *RectangularRippleBehavior* and ButtonBehavior and *FitImage* classes documentation.

class kivymd.uix.imagelist.imagelist.**MDSmartTileOverlayContainer**(**args*, ***kwargs*)

Implements a container for custom widgets to be added to the tile.

Changed in version 2.0.0: The *SmartTileOverlayBox* class has been renamed to *MDSmartTileOverlayContainer*.

For more information, see in the BoxLayout class documentation.

class kivymd.uix.imagelist.imagelist.**MDSmartTile**(***kwargs*)

A tile for more complex needs.

For more information, see in the *MDRelativeLayout* class documentation.

Includes an image, a container to place overlays and a box that can act as a header or a footer, as described in the Material Design specs.

Events

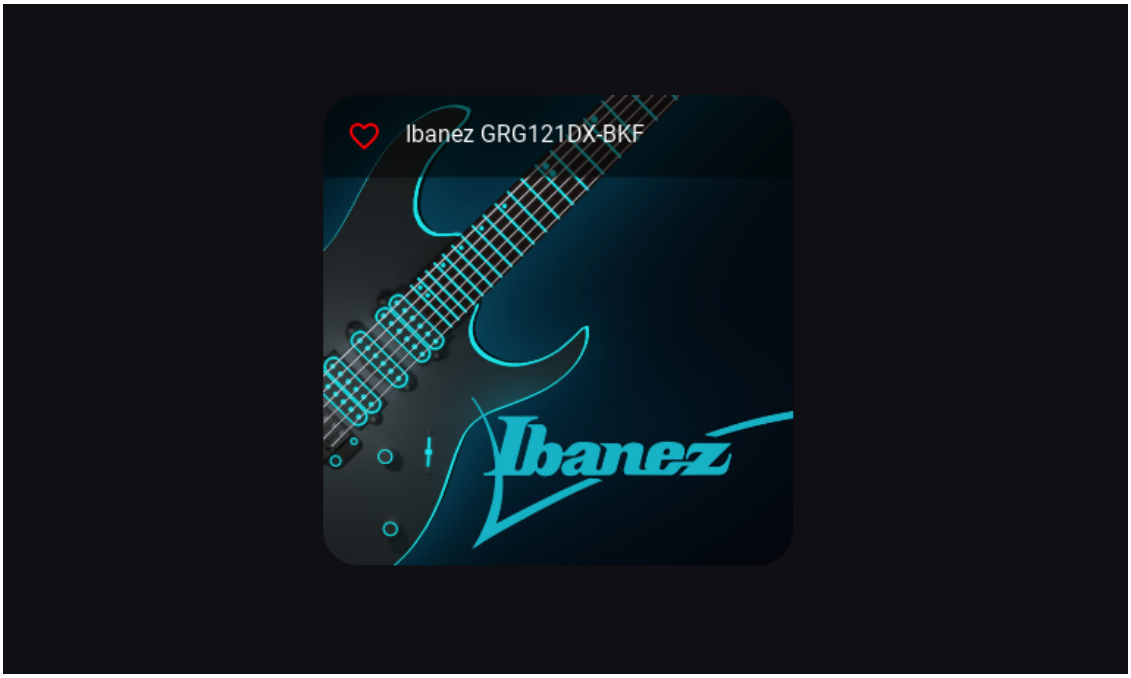
on_press
Fired when the button is pressed.

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

overlay_mode

Determines weather the information box acts as a header or footer to the image. Available are options: *'footer'*, *'header'*.

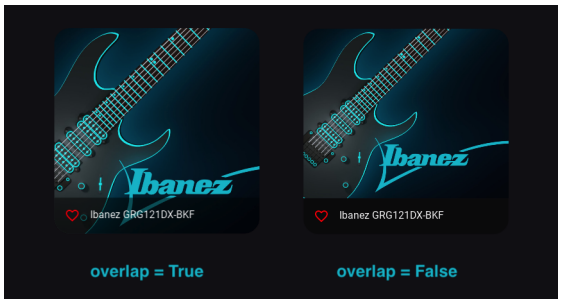
Changed in version 2.0.0: The *box_position* attribute has been renamed to *overlay_mode*.



overlay_mode is a OptionProperty and defaults to *'footer'*.

overlap

Determines if the *header/footer* overlaps on top of the image or not.



overlap is a BooleanProperty and defaults to *True*.

on_release(**args*)

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

on_press(**args*)

Fired when the button is pressed.

add_widget(*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)
```

2.3.39 Swiper

Usage

```
MDSwiper:

    MDSwiperItem:

    MDSwiperItem:

    MDSwiperItem:
```

Example

```
from kivy.lang.builder import Builder

from kivymd.app import MDApp

kv = '''
<MySwiper@MDSwiperItem>

    FitImage:
        source: "bg.jpg"
        radius: [dp(20),]


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

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

        MySwiper:

        MySwiper:

        MySwiper:

        MySwiper:

        MySwiper:
'''


class Main(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(kv)


Main().run()
```

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

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

MDSwiper provides the following events for use:

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

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

API - kivymd.uix.swiper.swiper

class kivymd.uix.swiper.swiper.**MDSwiperItem**(**args*, ***kwargs*)

Swiper item class.

For more information, see in the *MDBoxLayout* class documentation.

class kivymd.uix.swiper.swiper.**MDSwiper**(**args*, ***kwargs*)

Swiper class.

For more information, see in the *MDScrollView* 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

Wheter to swipe on mouse wheel scrolling or not.

swipe_on_scroll is an BooleanProperty and defaults to *True*.

add_widget(*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(*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(*index*)

Switch to given *MDSwiperItem* index.

get_current_index()

Returns the current *MDSwiperItem* index.

get_current_item()

Returns the current *MDSwiperItem* instance.

get_items()

Returns the list of *MDSwiperItem* children.

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

---

on_swipe()

on_pre_swipe()

on_overswipe_right()

on_overswipe_left()

on_swipe_left()

on_swipe_right()

swipe_left()

swipe_right()

on_scroll_start(*touch*, *check_children=True*)

on_touch_down(*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(*touch*)

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

See *on_touch_down()* for more information.

2.3.40 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: 4
            left_action_items: [['menu', lambda x: x]]

        MDScrollViewRefreshLayout:
            id: refresh_layout
            refresh_callback: app.refresh_callback
            root_layout: root
            spinner_color: "brown"
            circle_color: "white"

            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.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        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*)

Refresh layout class.

For more information, see in the *ThemableBehavior* and *MDScrollView* 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*.

spinner_color

Color of the spinner in (r, g, b, a) or string format.

New in version 1.2.0.

spinner_color is a ColorProperty and defaults to *[1, 1, 1, 1]*.

circle_color

Color of the ellipse around the spinner in (r, g, b, a) or string format.

New in version 1.2.0.

circle_color is a ColorProperty and defaults to *None*.

show_transition

Transition of the spinner's opening.

New in version 1.2.0.

show_transition is a StringProperty and defaults to *'out_elastic'*.

show_duration

Duration of the spinner display.

New in version 1.2.0.

show_duration is a NumericProperty and defaults to *0.8*.

hide_transition

Transition of hiding the spinner.

New in version 1.2.0.

hide_transition is a StringProperty and defaults to *'out_elastic'*.

hide_duration

Duration of hiding the spinner.

New in version 1.2.0.

hide_duration is a NumericProperty and defaults to *0.8*.

on_touch_up(**args*)

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

See on_touch_down() for more information.

refresh_done() → None

2.3.41 Tooltip

See also:

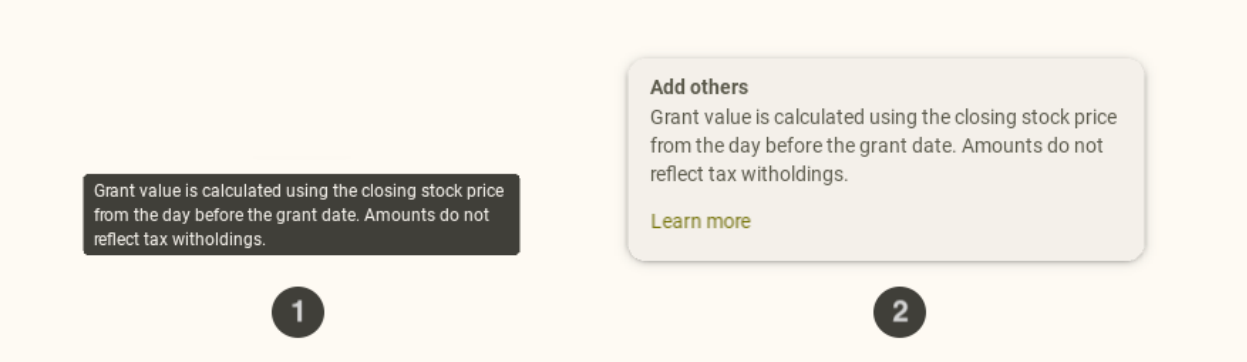
Material Design spec, Tooltips

Tooltips display brief labels or messages.

- Use tooltips to add additional context to a button or other UI element
- Two types: plain and rich
- Use plain tooltips to describe elements or actions of icon buttons
- Use rich tooltips to provide more details, like describing the value of a feature
- Rich tooltips can include an optional title, link, and buttons

KivyMD provides two types of tooltip:

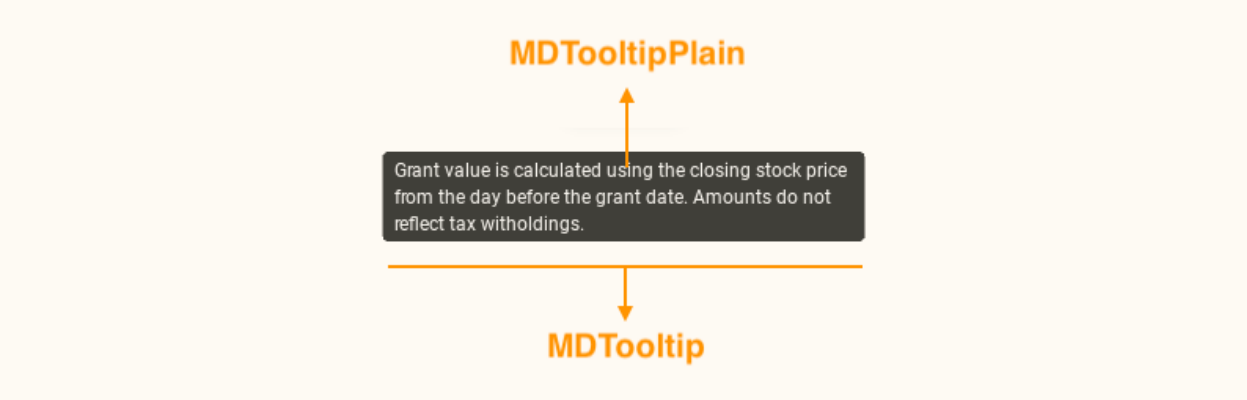
1. Plain tooltip
2. Rich tooltip



Usage of tooltip plain

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

from kivymd.uix.button import MDButton
from kivymd.uix.tooltip import MDTooltip
from kivymd.app import MDApp

KV = '''
<YourTooltipClass>

    MDTooltipPlain:
        text:
            "Grant value is calculated using the closing stock price \\n" \
            "from the day before the grant date. Amounts do not \\n" \
            "reflect tax witholdings."


<TooltipMDIconButton>

    MDButtonText:
        text: root.text


MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    TooltipMDIconButton:
        text: "Tooltip button"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class YourTooltipClass(MDTooltip):
    '''Implements your tooltip base class.'''


class TooltipMDIconButton(YourTooltipClass, MDButton):
    '''Implements a button with tooltip behavior.'''

    text = StringProperty()


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


Example().run()
```

The anatomy of a plain tooltip



Usage of tooltip rich

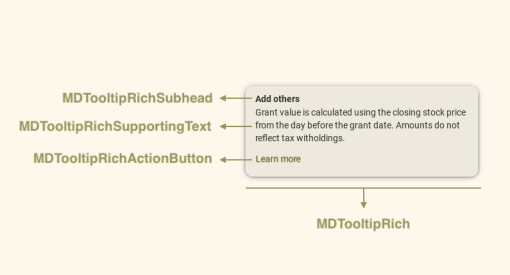
```
from kivy.lang import Builder
from kivy.properties import StringProperty

from kivymd.uix.button import MDButton
from kivymd.uix.tooltip import MDTooltip
from kivymd.app import MDApp

KV = '''
<YourTooltipClass>

    MDTooltipRich:
        id: tooltip
        auto_dismiss: False

        MDTooltipRichSubhead:
            text: "Add others"

        MDTooltipRichSupportingText:
            text:
                "Grant value is calculated using the closing stock price \\n" \
                "from the day before the grant date. Amounts do not \\n" \
                "reflect tax witholdings."

        MDTooltipRichActionButton:
            on_press: tooltip.dismiss()

            MDButtonText:
                text: "Learn more"


<TooltipMDIconButton>

    MDButtonText:
        text: root.text
```

(continues on next page)

(continued from previous page)

```
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    TooltipMDIconButton:
        text: "Tooltip button"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class YourTooltipClass(MDTooltip):
    '''Implements your tooltip base class.'''


class TooltipMDIconButton(YourTooltipClass, MDButton):
    '''Implements a button with tooltip behavior.'''

    text = StringProperty()


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


Example().run()
```

The anatomy of a plain tooltip



API - kivymd.uix.tooltip.tooltip

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

Tooltip class.

For more information, see in the *TouchBehavior* class documentation.

Events

***on_open*:**
Fired when the tooltip opens.

***on_dismiss*:**
Fired when the tooltip is closed.

tooltip_display_delay

Tooltip display delay.

Note: This property only works on desktop.

---

tooltip_display_delay is an BoundedNumericProperty and defaults to *0*, min of *0* & max of *4*.

shift_y

Y-offset of tooltip to the top.

shift_y is an NumericProperty and defaults to *0*.

shift_right

Shifting the tooltip to the right.

New in version 1.0.0.

shift_right is an NumericProperty and defaults to *0*.

shift_left

Shifting the tooltip to the left.

New in version 1.0.0.

shift_left is an NumericProperty and defaults to *0*.

delete_clock(*widget*, *touch*, **args*)

Removes a key event from *touch.ud*.

adjust_tooltip_position() → tuple

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

display_tooltip(**args*) → None

Adds a tooltip widget to the screen and animates its display.

animation_tooltip_show(**args*) → None

Animation of opening tooltip on the screen.

animation_tooltip_dismiss(**args*) → None

Animation of closing tooltip on the screen.

New in version 1.0.0.

remove_tooltip(**args*) → None

Removes the tooltip widget from the screen.

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

Add a new widget as a child of this widget.

on_long_touch(*touch*, **args*) → None

Fired when the widget is pressed for a long time.

on_enter(**args*) → None

Fired when mouse enter the bbox of the widget.

on_leave(**args*) → None

Fired when the mouse goes outside the widget border.

on_open() → None

Default display event handler.

Changed in version 2.0.0: Rename from *on_show* to *on_open*.

on_dismiss() → None

Default dismiss event handler.

New in version 1.0.0.

class kivymd.uix.tooltip.tooltip.**MDTooltipPlain**(**args*, ***kwargs*)

Tooltip plain class.

New in version 2.0.0.

For more information, see in the *MDLabel* and *ScaleBehavior* classes documentation.

class kivymd.uix.tooltip.tooltip.**MDTooltipRichSupportingText**(**args*, ***kwargs*)

Implements supporting text for the *MDTooltipRich* class.

New in version 2.0.0.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.tooltip.tooltip.**MDTooltipRichSubhead**(**args*, ***kwargs*)

Implements subhead text for the *MDTooltipRich* class.

New in version 2.0.0.

For more information, see in the *MDLabel* class documentation.

class kivymd.uix.tooltip.tooltip.**MDTooltipRichActionButton**(**args*, ***kwargs*)

Implements action button for the *MDTooltipRich* class.

New in version 2.0.0.

For more information, see in the *MDButton* class documentation.

on_enter() → None

Fired when mouse enter the bbox of the widget.

on_leave() → None

Fired when the mouse goes outside the widget border.

class kivymd.uix.tooltip.tooltip.**MDTooltipRich**(**args*, ***kwargs*)

Tooltip rich class.

New in version 2.0.0.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and *CommonElevationBehavior* and *ScaleBehavior* and *StateLayerBehavior* and BoxLayout and classes documentation.

auto_dismiss

This property determines if the view is automatically dismissed when the cursor goes outside of the tooltip body.

auto_dismiss is a BooleanProperty and defaults to True.

on_leave() → None

Fired when the mouse goes outside the widget border.

dismiss() → None

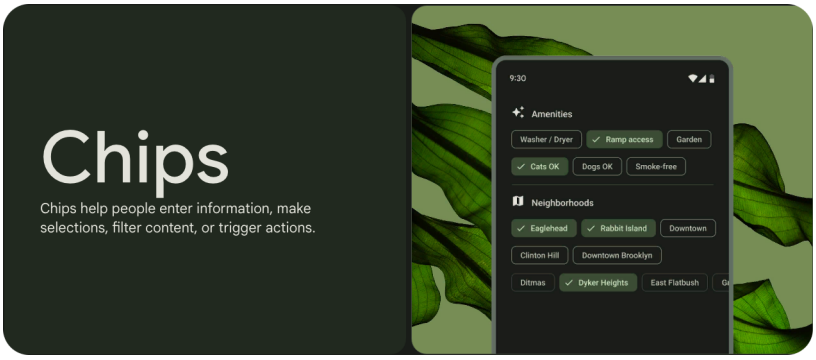
Hides the tooltip.

2.3.42 Chip

See also:

Material Design 3 spec, Chips

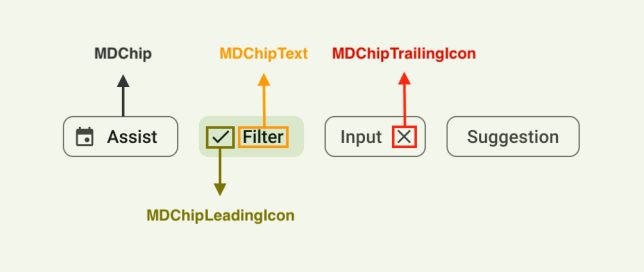
Chips can show multiple interactive elements together in the same area, such as a list of selectable movie times, or a series of email contacts. There are four types of chips: assist, filter, input, and suggestion.



Usage

```
MDChip:

    MDChipLeadingAvatar:  # MDChipLeadingIcon

    MDChipText:

    MDChipTrailingIcon:
```

Anatomy



Example

Declarative KV style

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDChip:
        pos_hint: {"center_x": .5, "center_y": .5}

        MDChipText:
            text: "MDChip"
'''


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


Example().run()
```

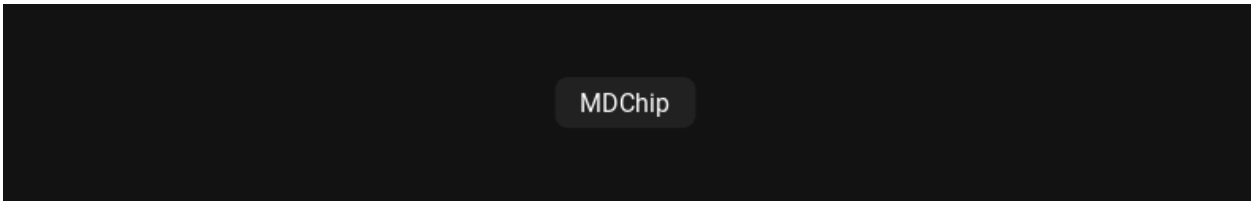
Declarative Python style

```
from kivymd.app import MDApp
from kivymd.uix.chip import MDChip, MDChipText
from kivymd.uix.screen import MDScreen


class Example(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return (
            MDScreen(
                MDChip(
                    MDChipText(
```

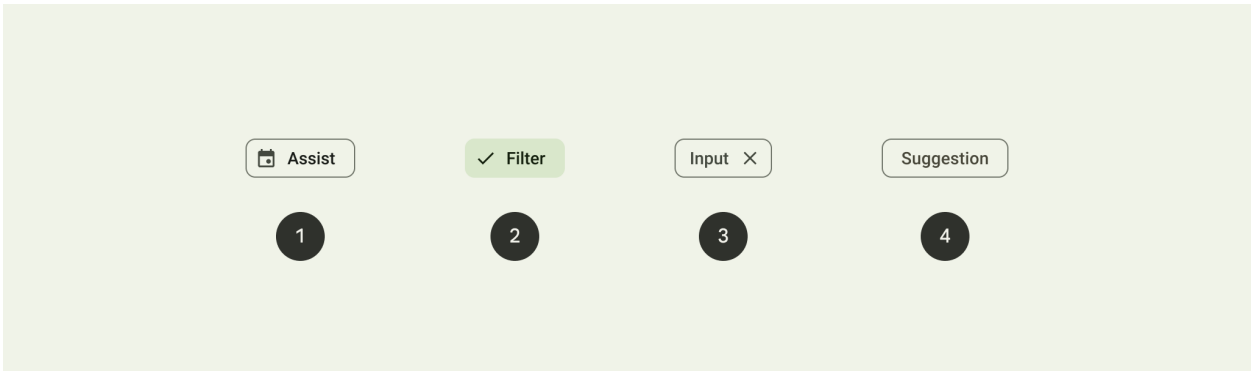
(continues on next page)

(continued from previous page)

```
                        text="MDChip"
                    ),
                    pos_hint={"center_x": .5, "center_y": .5},
                )
            )
        )


Example().run()
```



The following types of chips are available:

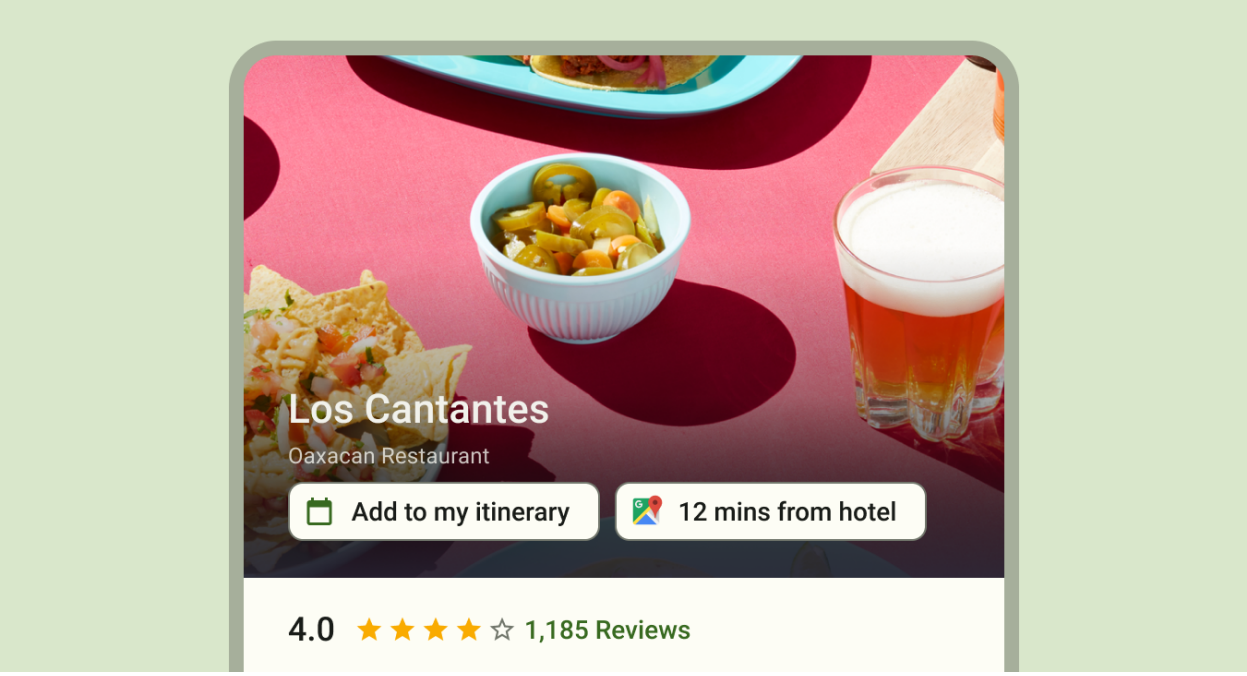


- *Assist*
- *Filter*
- *Input*
- *Suggestion*

Assist

Assist chips represent smart or automated actions that can span multiple apps, such as opening a calendar event from the home screen. Assist chips function as though the user asked an assistant to complete the action. They should appear dynamically and contextually in a UI.

An alternative to assist chips are buttons, which should appear persistently and consistently.



Example of assist

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
<CommonLabel@MDLabel>
    adaptive_size: True
    theme_text_color: "Custom"
    text_color: "#e6e9df"


<CommonAssistChip@MDChip>
    # Custom attribute.
    text: ""
    icon: ""

    # Chip attribute.
    type: "assist"
    md_bg_color: "#2a3127"
    line_color: "grey"
    elevation: 1
    shadow_softness: 2

    MDChipLeadingIcon:
        icon: root.icon
        theme_text_color: "Custom"
        text_color: "#68896c"
```

(continues on next page)

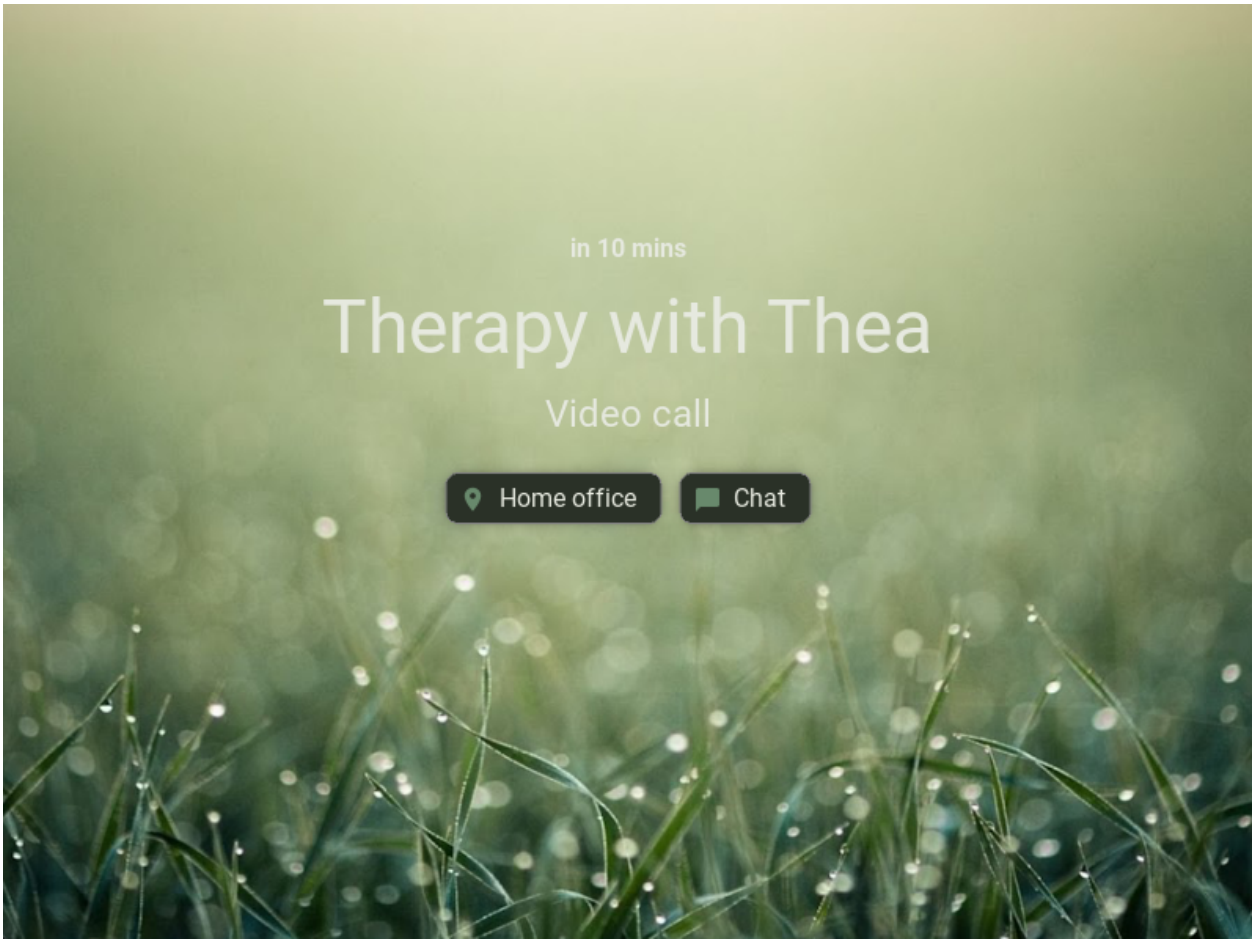
(continued from previous page)

```
    MDChipText:
        text: root.text
        theme_text_color: "Custom"
        text_color: "#e6e9df"


MDScreen:

    FitImage:
        source: "bg.png"

    MDBoxLayout:
        orientation: "vertical"
        adaptive_size: True
        pos_hint: {"center_y": .6, "center_x": .5}

        CommonLabel:
            text: "in 10 mins"
            bold: True
            pos_hint: {"center_x": .5}

        CommonLabel:
            text: "Therapy with Thea"
            font_style: "H3"
            padding_y: "12dp"

        CommonLabel:
            text: "Video call"
            font_style: "H5"
            pos_hint: {"center_x": .5}

        MDBoxLayout:
            adaptive_size: True
            pos_hint: {"center_x": .5}
            spacing: "12dp"
            padding: 0, "24dp", 0, 0

            CommonAssistChip:
                text: "Home office"
                icon: "map-marker"

            CommonAssistChip:
                text: "Chat"
                icon: "message"

        MDWidget:
'''


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

(continues on next page)

(continued from previous page)

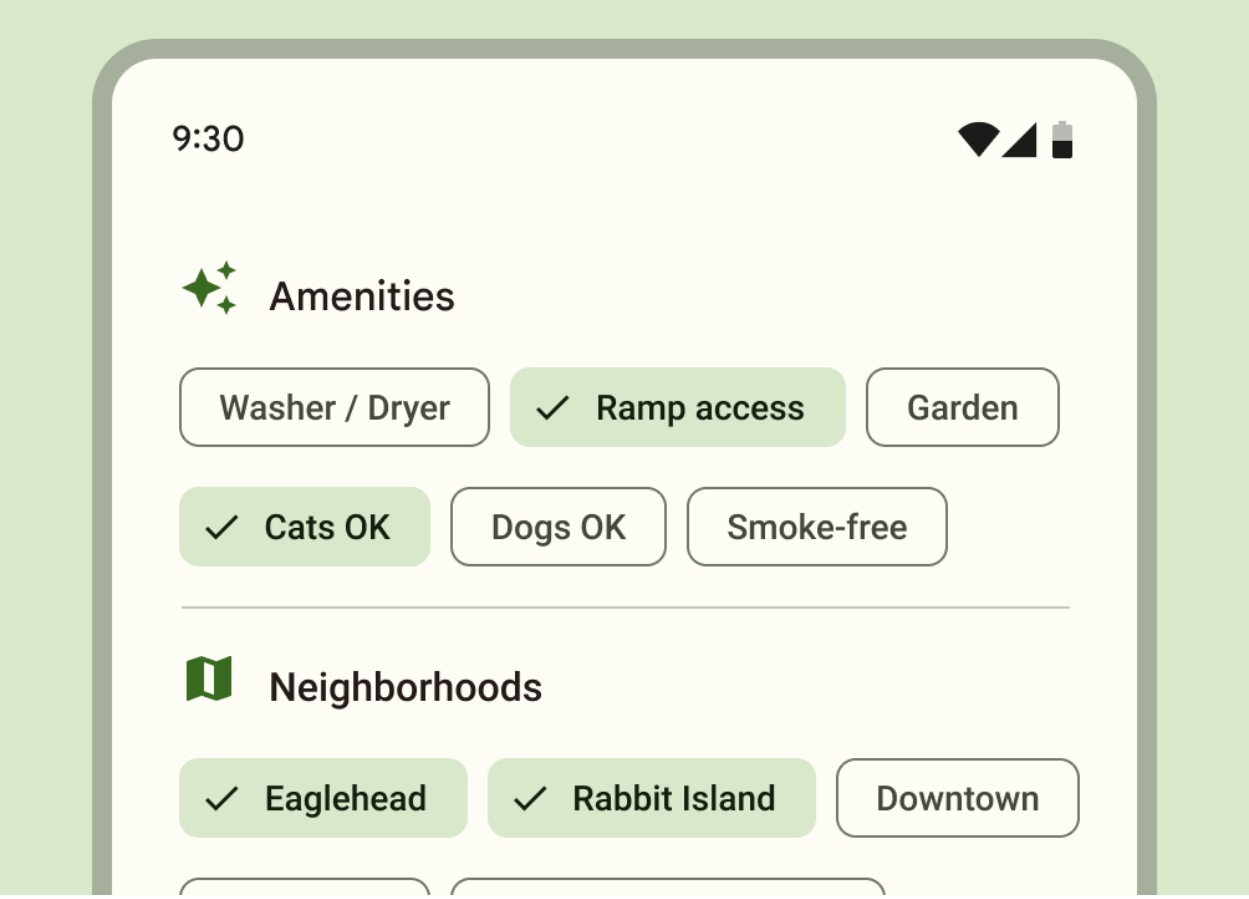
```
        self.theme_cls.theme_style = "Dark"
        return Builder.load_string(KV)


Example().run()
```



Filter

Filter chips use tags or descriptive words to filter content. They can be a good alternative to toggle buttons or checkboxes.

Tapping on a filter chip activates it and appends a leading checkmark icon to the starting edge of the chip label.



Example of filtering

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

from kivymd.app import MDApp
from kivymd.uix.chip import MDChip, MDChipText
from kivymd.uix.list import MDListItem
from kivymd.icon_definitions import md_icons
from kivymd.uix.screen import MDScreen

import asynckivy

Builder.load_string(
    '''
<CustomOneLineIconListItem>

    MDListItemLeadingIcon:
        icon: root.icon

    MDListItemHeadlineText:
        text: root.text
```

(continues on next page)

(continued from previous page)

```
<PreviewIconsScreen>

    MDBoxLayout:
        orientation: "vertical"
        spacing: "14dp"
        padding: "20dp"

        MDTextField:
            id: search_field
            mode: "outlined"
            on_text: root.set_list_md_icons(self.text, True)

            MDTextFieldLeadingIcon:
                icon: "magnify"

            MDTextFieldHintText:
                text: "Search icon"

        MDBoxLayout:
            id: chip_box
            spacing: "12dp"
            adaptive_height: True

        RecycleView:
            id: rv
            viewclass: "CustomOneLineIconListItem"
            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(MDListItem):
    icon = StringProperty()
    text = StringProperty()


class PreviewIconsScreen(MDScreen):
    filter = ListProperty()  # list of tags for filtering icons

    def set_filter_chips(self):
        '''Asynchronously creates and adds chips to the container.'''

        async def set_filter_chips():
```

(continues on next page)

(continued from previous page)

```
            for tag in ["Outline", "Off", "On"]:
                await asynckivy.sleep(0)
                chip = MDChip(
                    MDChipText(
                        text=tag,
                    ),
                    type="filter",
                    md_bg_color="#303A29",
                )
                chip.bind(active=lambda x, y, z=tag: self.set_filter(y, z))
                self.ids.chip_box.add_widget(chip)

        asynckivy.start(set_filter_chips())

    def set_filter(self, active: bool, tag: str) -> None:
        '''Sets a list of tags for filtering icons.'''

        if active:
            self.filter.append(tag)
        else:
            self.filter.remove(tag)

    def set_list_md_icons(self, text="", search=False) -> None:
        '''Builds a list of icons.'''

        def add_icon_item(name_icon):
            self.ids.rv.data.append(
                {
                    "icon": name_icon,
                    "text": name_icon,
                }
            )

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


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = PreviewIconsScreen()

    def build(self) -> PreviewIconsScreen:
        self.theme_cls.theme_style = "Dark"
        return self.screen
```

(continues on next page)

(continued from previous page)

```
    def on_start(self) -> None:
        self.screen.set_list_md_icons()
        self.screen.set_filter_chips()


Example().run()
```

Tap a chip to select it. Multiple chips can be selected or unselected:

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.chip import MDChip, MDChipText
from kivymd.uix.screen import MDScreen

import asynckivy

Builder.load_string(
    '''
<ChipScreen>

    MDBoxLayout:
        orientation: "vertical"
        spacing: "14dp"
        padding: "20dp"

        MDLabel:
            adaptive_height: True
            text: "Select Type"

        MDStackLayout:
            id: chip_box
            spacing: "12dp"
            adaptive_height: True

        MDWidget:

    MDButton:
        pos: "20dp", "20dp"
        on_release: root.unchecks_chips()

        MDButtonText:
            text: "Uncheck chips"
    '''
)


class ChipScreen(MDScreen):
    async def create_chips(self):
        '''Asynchronously creates and adds chips to the container.'''
```

(continues on next page)

(continued from previous page)

```
        for tag in ["Extra Soft", "Soft", "Medium", "Hard"]:
            await asynckivy.sleep(0)
            self.ids.chip_box.add_widget(
                MDChip(
                    MDChipText(
                        text=tag,
                    ),
                    type="filter",
                    md_bg_color="#303A29",
                    active=True,
                )
            )

    def unchecks_chips(self) -> None:
        '''Removes marks from all chips.'''

        for chip in self.ids.chip_box.children:
            if chip.active:
                chip.active = False


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = ChipScreen()

    def build(self) -> ChipScreen:
        self.theme_cls.theme_style = "Dark"
        return self.screen

    def on_start(self) -> None:
        asynckivy.start(self.screen.create_chips())


Example().run()
```

Alternatively, a single chip can be selected. This offers an alternative to toggle buttons, radio buttons, or single select menus:

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.chip import MDChip, MDChipText
from kivymd.uix.screen import MDScreen

import asynckivy

Builder.load_string(
    '''
<ChipScreen>
```

(continues on next page)

(continued from previous page)

```
    MDBoxLayout:
        orientation: "vertical"
        spacing: "14dp"
        padding: "20dp"

        MDLabel:
            adaptive_height: True
            text: "Select Type"

        MDStackLayout:
            id: chip_box
            spacing: "12dp"
            adaptive_height: True

        MDWidget:
    '''
)


class ChipScreen(MDScreen):
    async def create_chips(self):
        '''Asynchronously creates and adds chips to the container.'''

        for tag in ["Extra Soft", "Soft", "Medium", "Hard"]:
            await asynckivy.sleep(0)
            chip = MDChip(
                MDChipText(
                    text=tag,
                ),
                type="filter",
                md_bg_color="#303A29",

            )
            chip.bind(active=self.uncheck_chip)
            self.ids.chip_box.add_widget(chip)

    def uncheck_chip(self, current_chip: MDChip, active: bool) -> None:
        '''Removes a mark from an already marked chip.'''

        if active:
            for chip in self.ids.chip_box.children:
                if current_chip is not chip:
                    if chip.active:
                        chip.active = False


class Example(MDApp):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.screen = ChipScreen()
```

(continues on next page)

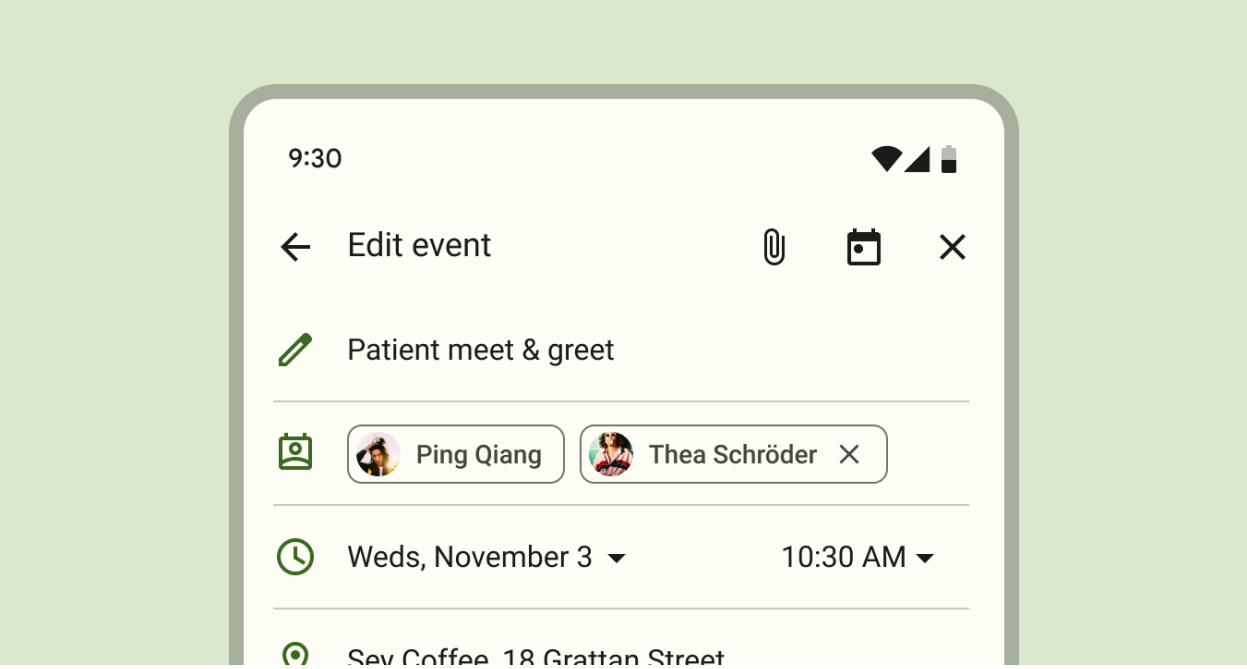
(continued from previous page)

```
    def build(self) -> ChipScreen:
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "LightGreen"
        return self.screen

    def on_start(self) -> None:
        asynckivy.start(self.screen.create_chips())


Example().run()
```

Input

Input chips represent discrete pieces of information entered by a user, such as Gmail contacts or filter options within a search field.

They enable user input and verify that input by converting text into chips.



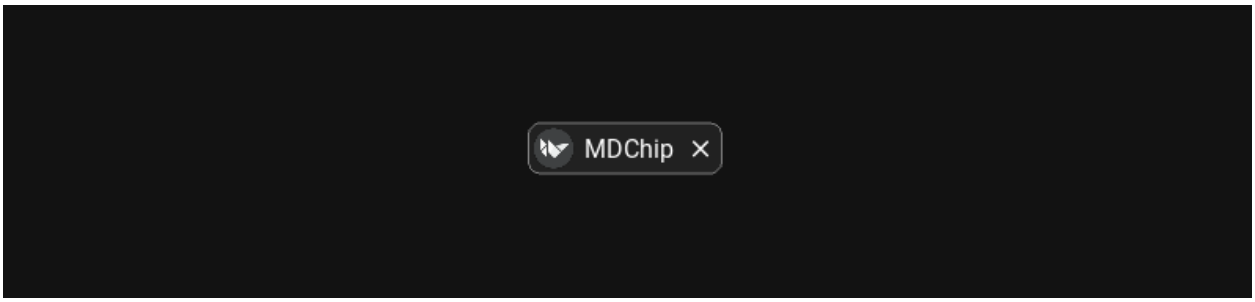
Example of input

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
```

(continues on next page)

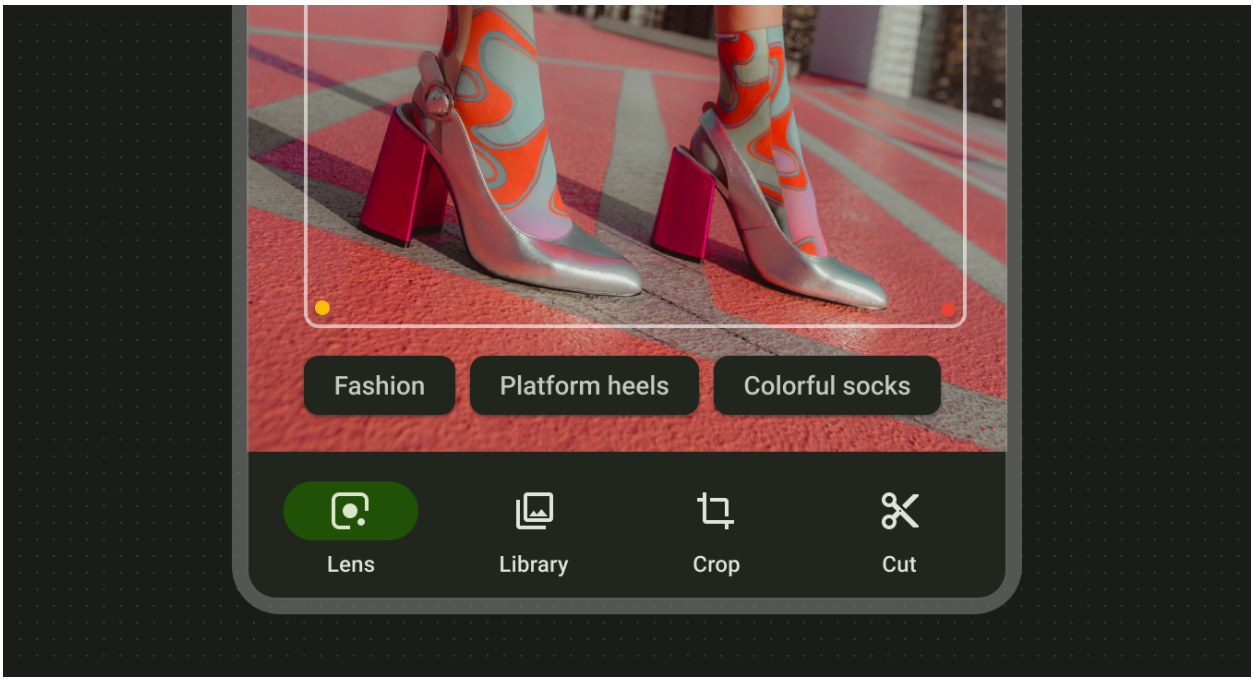
(continued from previous page)

```
    MDChip:
        pos_hint: {"center_x": .5, "center_y": .5}
        type: "input"
        line_color: "grey"
        _no_ripple_effect: True

        MDChipLeadingAvatar:
            source: "data/logo/kivy-icon-128.png"

        MDChipText:
            text: "MDChip"

        MDChipTrailingIcon:
            icon: "close"
'''


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


Example().run()
```



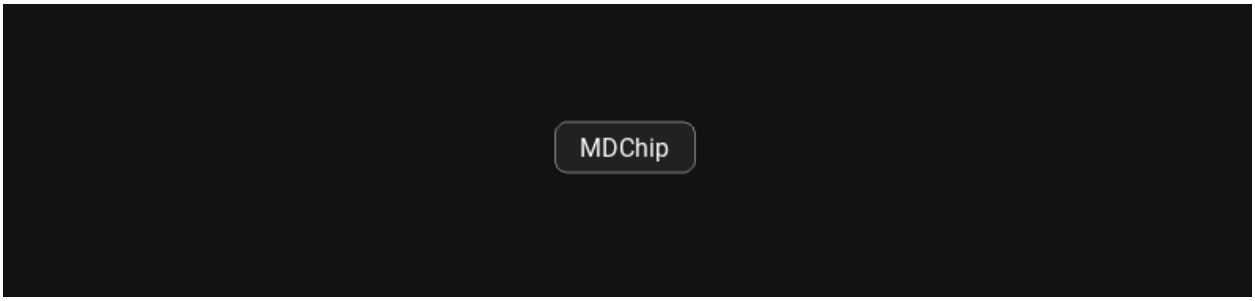
Suggestion

Suggestion chips help narrow a user's intent by presenting dynamically generated suggestions, such as possible responses or search filters.



Example of suggestion

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDChip:
        pos_hint: {"center_x": .5, "center_y": .5}
        type: "suggestion"
        line_color: "grey"

        MDChipText:
            text: "MDChip"
'''


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


Example().run()
```



API break

1.2.0 version

```
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()
```

2.0.0 version

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDChip:
        pos_hint: {"center_x": .5, "center_y": .5}
        line_color: "grey"
```

(continues on next page)

(continued from previous page)

```
        on_release: app.on_release_chip(self)

        MDChipText:
            text: "MDChip"
'''


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

    def on_release_chip(self, instance_check):
        print(instance_check)


Example().run()
```

API - kivymd.uix.chip.chip

class kivymd.uix.chip.chip.**MDChipLeadingAvatar**(***kwargs*)

Implements the leading avatar for the chip.

For more information, see in the *CircularRippleBehavior* and *ScaleBehavior* and ButtonBehavior and *MDIcon* classes documentation.

class kivymd.uix.chip.chip.**MDChipLeadingIcon**(***kwargs*)

Implements the leading icon for the chip.

For more information, see in the *CircularRippleBehavior* and *ScaleBehavior* and ButtonBehavior and *MDIcon* classes documentation.

class kivymd.uix.chip.chip.**MDChipTrailingIcon**(***kwargs*)

Implements the trailing icon for the chip.

For more information, see in the *CircularRippleBehavior* and *ScaleBehavior* and ButtonBehavior and *MDIcon* classes documentation.

class kivymd.uix.chip.chip.**MDChipText**(**args*, ***kwargs*)

Implements the label for the chip.

For more information, see in the *MDLabel* classes documentation.

text_color_disabled

The text color in (r, g, b, a) or string format of the chip when the chip is disabled.

New in version 2.0.0.

text_color_disabled is a ColorProperty and defaults to *None*.

class kivymd.uix.chip.chip.**MDChip**(**args*, ***kwargs*)

Chip class.

For more information, see in the *MDBoxLayout* and *RectangularRippleBehavior* and ButtonBehavior and *CommonElevationBehavior* and *TouchBehavior* classes documentation.

radius

Chip radius.

radius is an VariableListProperty and defaults to *[dp(8), dp(8), dp(8), dp(8)]*.

type

Type of chip.

New in version 2.0.0.

Available options are: *'assist'*, *'filter'*, *'input'*, *'suggestion'*.

type is an OptionProperty and defaults to *'suggestion'*.

active

Whether the check is marked or not.

New in version 1.0.0.

active is an BooleanProperty and defaults to *False*.

selected_color

The background color of the chip in the marked state in (r, g, b, a) or string format.

New in version 2.0.0.

selected_color is an ColorProperty and defaults to *None*.

line_color_disabled

The color of the outline in the disabled state

New in version 2.0.0.

line_color_disabled is an ColorProperty and defaults to *None*.

on_long_touch(**args*) → None

Fired when the widget is pressed for a long time.

on_line_color(*instance*, *value*) → None

Fired when the values of `line_color` change.

on_type(*instance*, *value: str*) → None

Fired when the values of *type* change.

on_active(*instance_check*, *active_value: bool*) → None

Called when the values of *active* change.

complete_anim_ripple(**args*) → None

Called at the end of the ripple animation.

remove_marked_icon_from_chip() → None

add_marked_icon_to_chip() → None

Adds and animates a check icon to the chip.

set_chip_bg_color(*color: list | str*) → None

Animates the background color of the chip.

on_press(**args*) → None

Fired when the button is pressed.

`on_release`(**args*) → None

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

`add_widget`(*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.

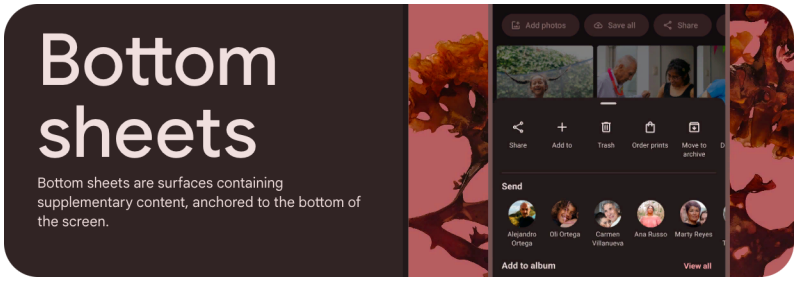
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.43 BottomSheet

See also:

Material Design spec, Sheets: bottom

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



Usage

```
Root:

    MDNavigationLayout:

        MDScreenManager:

            [...]

        MDBottomSheet:
```

The bottom sheet has two types:

- *Standard*
- *Modal*

Standard

Standard bottom sheets co-exist with the screen's main UI region and allow for simultaneously viewing and interacting with both regions, especially when the main UI region is frequently scrolled or panned.

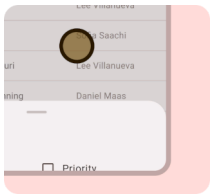
Use a standard bottom sheet to display content that complements the screen's primary content, such as an audio player in a music app.

Standard bottom sheets are elevated above the main UI region so their visibility is not affected by panning or scrolling.

Modal

Like dialogs, modal bottom sheets appear in front of app content, disabling all other app functionality when they appear, and remaining on screen until confirmed, dismissed, or a required action has been taken.

Tapping the scrim dismisses a modal bottom sheet.



Add elements to `MDBottomSheetDragHandleTitle` class

```
MDBottomSheet:

    MDBottomSheetDragHandle:

        MDBottomSheetDragHandleTitle:
            text: "MDBottomSheet"
            adaptive_height: True
            pos_hint: {"center_y": .5}

        MDBottomSheetDragHandleButton:
            icon: "close"
```

A practical example with standard bottom sheet

(A double tap on the map to open the bottom sheet)

```
import asynckivy

from kivy.lang import Builder
from kivy.properties import StringProperty, ObjectProperty, BooleanProperty
from kivy_garden.mapview import MapView

from kivymd.app import MDApp
from kivymd.uix.behaviors import TouchBehavior
from kivymd.uix.boxlayout import MDBoxLayout

KV = '''
#:import MapSource kivy_garden.mapview.MapSource
#:import asynckivy asynckivy


<TypeMapElement>
```

(continues on next page)

(continued from previous page)

```
    orientation: "vertical"
    adaptive_height: True
    spacing: "8dp"

    MDIconButton:
        id: icon
        icon: root.icon
        theme_bg_color: "Custom"
        md_bg_color: "#EDF1F9" if not root.selected else app.theme_cls.primaryColor
        pos_hint: {"center_x": .5}
        theme_icon_color: "Custom"
        icon_color: "white" if root.selected else "black"
        on_release: app.set_active_element(root, root.title.lower())

    MDLabel:
        text: root.title
        pos_hint: {"center_x": .5}
        halign: "center"
        adaptive_height: True


MDScreen:

    MDNavigationLayout:

        MDScreenManager:

            MDScreen:

                CustomMapView:
                    bottom_sheet: bottom_sheet
                    map_source: MapSource(url=app.map_sources[app.current_map])
                    lat: 46.5124
                    lon: 47.9812
                    zoom: 12

        MDBottomSheet:
            id: bottom_sheet
            sheet_type: "standard"
            size_hint_y: None
            height: "150dp"
            on_open: asynckivy.start(app.generate_content())

            MDBottomSheetDragHandle:
                drag_handle_color: "grey"

                MDBottomSheetDragHandleTitle:
                    text: "Select type map"
                    pos_hint: {"center_y": .5}

                MDBottomSheetDragHandleButton:
                    icon: "close"
```

(continues on next page)

(continued from previous page)

```
                    ripple_effect: False
                    on_release: bottom_sheet.set_state("toggle")

            BoxLayout:
                id: content_container
                padding: 0, 0, 0, "16dp"
'''


class TypeMapElement(MDBoxLayout):
    selected = BooleanProperty(False)
    icon = StringProperty()
    title = StringProperty()


class CustomMapView(MapView, TouchBehavior):
    bottom_sheet = ObjectProperty()

    def on_double_tap(self, touch, *args):
        if self.bottom_sheet:
            self.bottom_sheet.set_state("toggle")


class Example(MDApp):
    map_sources = {
        "street": "https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}",
        "sputnik": "https://mt1.google.com/vt/lyrs=s&x={x}&y={y}&z={z}",
        "hybrid": "https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}",
    }
    current_map = StringProperty("street")

    async def generate_content(self):
        icons = {
            "street": "google-street-view",
            "sputnik": "space-station",
            "hybrid": "map-legend",
        }
        if not self.root.ids.content_container.children:
            for i, title in enumerate(self.map_sources.keys()):
                await asynckivy.sleep(0)
                self.root.ids.content_container.add_widget(
                    TypeMapElement(
                        title=title.capitalize(),
                        icon=icons[title],
                        selected=not i,
                    )
                )

    def set_active_element(self, instance, type_map):
        for element in self.root.ids.content_container.children:
            if instance == element:
                element.selected = True
```

(continues on next page)

(continued from previous page)

```
                self.current_map = type_map
            else:
                element.selected = False

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

API break

1.2.0 version

```
Root:

    MDBottomSheet:

        # Optional.
        MDBottomSheetDragHandle:

            # Optional.
            MDBottomSheetDragHandleTitle:

            # Optional.
            MDBottomSheetDragHandleButton:

        MDBottomSheetContent:
            [...]
```

2.0.0 version

```
Root:

    MDNavigationLayout:

        MDScreenManager:

            # Your screen.
            MDScreen:

        MDBottomSheet:

            # Optional.
            MDBottomSheetDragHandle:

                # Optional.
                MDBottomSheetDragHandleTitle:
```

(continues on next page)

(continued from previous page)

```
            # Optional.
            MDBottomSheetDragHandleButton:
                icon: "close"

        # Your content.
        BoxLayout:
```

API - kivymd.uix.bottomsheet.bottomsheet

class kivymd.uix.bottomsheet.bottomsheet.**MDBottomSheetDragHandleButton**(***kwargs*)

Implements a close button (or other functionality) for the *MDBottomSheetDragHandle* container.

For more information, see in the *MDIconButton* class documentation.

New in version 1.2.0.

class kivymd.uix.bottomsheet.bottomsheet.**MDBottomSheetDragHandleTitle**(**args*, ***kwargs*)

Implements a header for the *MDBottomSheetDragHandle* container.

For more information, see in the *MDLabel* class documentation.

New in version 1.2.0.

class kivymd.uix.bottomsheet.bottomsheet.**MDBottomSheetDragHandle**(***kwargs*)

Implements a container that can place the header of the bottom sheet and the close button. Also implements the event of dragging the bottom sheet on the parent screen.

For more information, see in the BoxLayout class documentation.

New in version 1.2.0.

drag_handle_color

Color of drag handle element in (r, g, b, a) or string format.

```
MDBottomSheet:

    MDBottomSheetDragHandle:
        drag_handle_color: "white"
```

drag_handle_color is an ColorProperty and defaults to *None*.

add_widget(*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)
```

class kivymd.uix.bottomsheet.bottomsheet.**MDBottomSheet**(**args*, ***kwargs*)

Bottom sheet class.

For more information, see in the *MDNavigationDrawer* class documentation.

sheet_type

Type of sheet.

Standard bottom sheets co-exist with the screen's main UI region and allow for simultaneously viewing and interacting with both regions, especially when the main UI region is frequently scrolled or panned. Use a standard bottom sheet to display content that complements the screen's primary content, such as an audio player in a music app.

Like dialogs, modal bottom sheets appear in front of app content, disabling all other app functionality when they appear, and remaining on screen until confirmed, dismissed, or a required action has been taken.

Changed in version 2.0.0: Rename from *type* to *sheet_type*.

sheet_type is a OptionProperty and defaults to *'modal'*.

on_sheet_type(*instance*, *value*) → None

Fired when the *sheet_type* value changes.

add_widget(*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_touch_move`(*touch*)

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

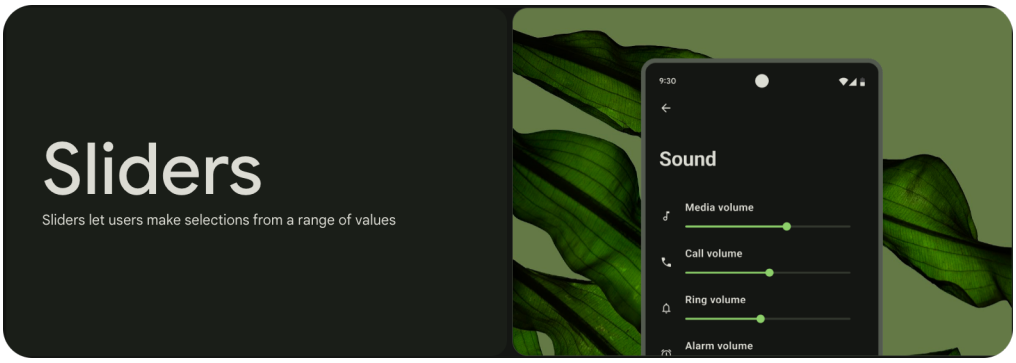
See `on_touch_down()` for more information.

2.3.44 Slider

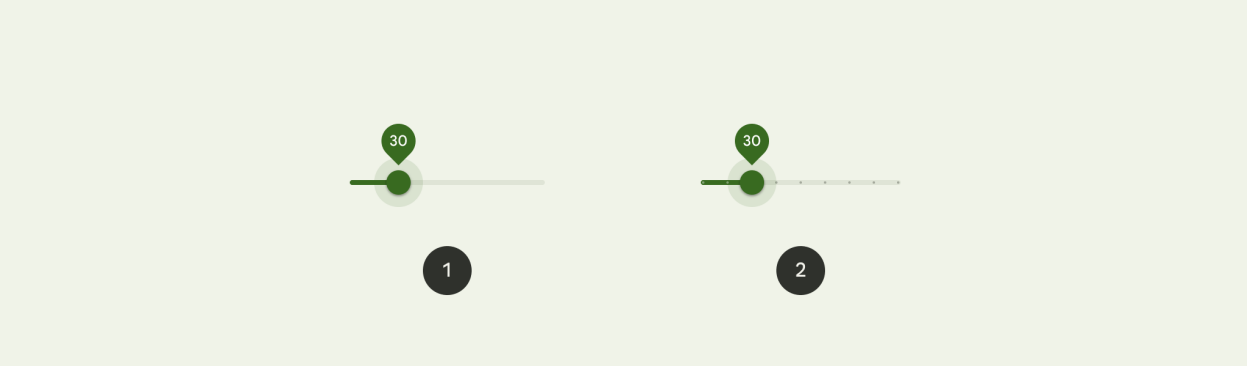
See also:

Material Design spec, Sliders

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



- Sliders should present the full range of choices that are available
- Two types: continuous and discrete
- The slider should immediately reflect any input made by a user



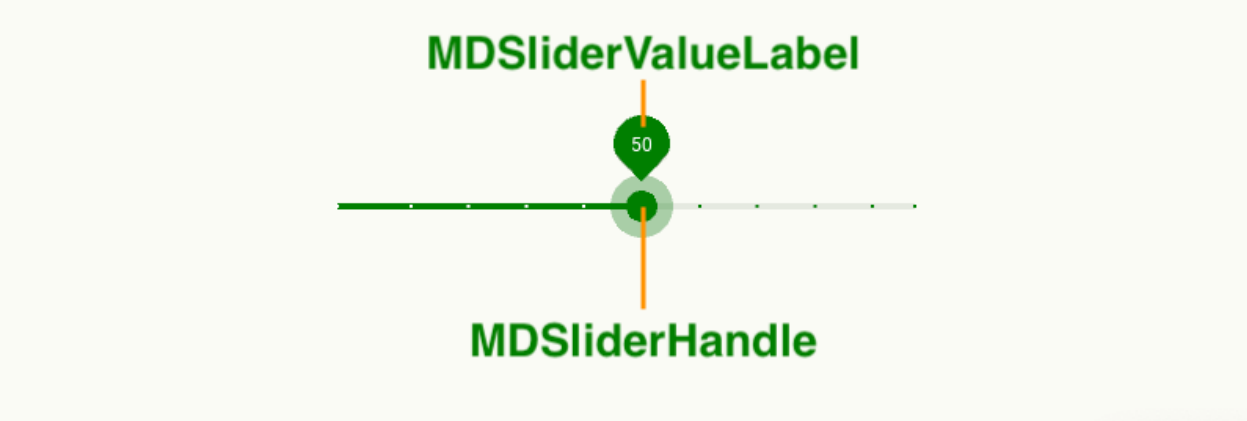
1. Continuous slider
2. Discrete slider

Usage

```
MDSlider(
    MDSliderHandle(
    ),
    MDSliderValueLabel(
    ),
    step=10,
    value=50,
)
```

```
MDSlider:
    step: 10
    value: 50

    MDSliderHandle:

    MDSliderValueLabel:
```

Anatomy



API - kivymd.uix.slider.slider

class kivymd.uix.slider.slider.**MDSlider**(**args*, ***kwargs*)

Slider class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and Slider classes documentation.

track_active_width

Width of the active track.

New in version 2.0.0.

track_active_width is an NumericProperty and defaults to *dp(4)*.

track_inactive_width

Width of the inactive track.

New in version 2.0.0.

track_inactive_width is an NumericProperty and defaults to *dp(4)*.

step_point_size

Step point size.

New in version 2.0.0.

step_point_size is an NumericProperty and defaults to *dp(1)*.

track_active_color

Color of the active track.

New in version 2.0.0.

Changed in version 2.0.0: Rename from *track_color_active* to *track_active_color*

track_active_color is an ColorProperty and defaults to *None*.

track_active_step_point_color

Color of step points on active track.

New in version 2.0.0.

track_active_step_point_color is an ColorProperty and defaults to *None*.

track_inactive_step_point_color

Color of step points on inactive track.

New in version 2.0.0.

track_inactive_step_point_color is an ColorProperty and defaults to *None*.

track_inactive_color

Color of the inactive track.

New in version 2.0.0.

Changed in version 2.0.0: Rename from *track_color_inactive* to *track_inactive_color*

track_active_color is an ColorProperty and defaults to *None*.

value_container_show_anim_duration

Duration of the animation opening of the label value.

New in version 2.0.0.

value_container_show_anim_duration is an NumericProperty and defaults to *0.2*.

value_container_hide_anim_duration

Duration of closing the animation of the label value.

New in version 2.0.0.

value_container_hide_anim_duration is an NumericProperty and defaults to *0.2*.

value_container_show_anim_transition

The type of the opening animation of the label value.

New in version 2.0.0.

value_container_show_anim_transition is an StringProperty and defaults to *'out_circ'*.

value_container_hide_anim_transition

The type of the closing animation of the label value.

New in version 2.0.0.

value_container_hide_anim_transition is an StringProperty and defaults to *'out_circ'*.

handle_anim_transition

Handle animation type.

New in version 2.0.0.

handle_anim_transition is an StringProperty and defaults to *'out_circ'*.

handle_anim_duration

Handle animation duration.

New in version 2.0.0.

handle_anim_duration is an NumericProperty and defaults to *0.2*.

add_widget(*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)
```

update_points(*instance*, *step*) → None

Draws the step points on the slider.

on_size(**args*) → None

Fired when the widget is resized.

on_touch_down(*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_value_pos(**args*) → None

Fired when the *value_pos* value changes. Sets a new value for the value label texture.

on_touch_up(*touch*)

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

See *on_touch_down()* for more information.

on_touch_move(*touch*)

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

See *on_touch_down()* for more information.

on_handle_enter() → None

Scales the container of the label value.

on_handle_leave() → None

Scales the container of the label value.

class kivymd.uix.slider.slider.**MDSliderHandle**(***kwargs*)

Handle class.

New in version 2.0.0.

For more information, see in the *ThemableBehavior* and *BackgroundColorBehavior* and *FocusBehavior* and Widget classes documentation.

radius

Handle radius.

radius is an VariableListProperty and defaults to *[dp(10), dp(10), dp(10), dp(10)]*.

size

Handle size.

size is an ListProperty and defaults to *[dp(20), dp(20)]*.

state_layer_size

Handle state layer size.

state_layer_size is an ListProperty and defaults to *[dp(40), dp(40)]*.

state_layer_color

Handle state layer color.

state_layer_color is an ColorProperty and defaults to *None*.

on_enter() → None

Fired when mouse enter the bbox of the widget. Animates the display of the slider handle layer.

on_leave() → None

Fired when the mouse goes outside the widget border. Animates the hiding of the slider handle layer.

class kivymd.uix.slider.slider.**MDSliderValueLabel**(**args*, ***kwargs*)

Implements the value label.

For more information, see in the *MDLabel* class documentation.

New in version 2.0.0.

size

Container size for the label value.

handle_anim_transition is an ListProperty and defaults to *[dp(36), dp(36)]*.

2.3.45 SliverAppbar

New in version 1.0.0.

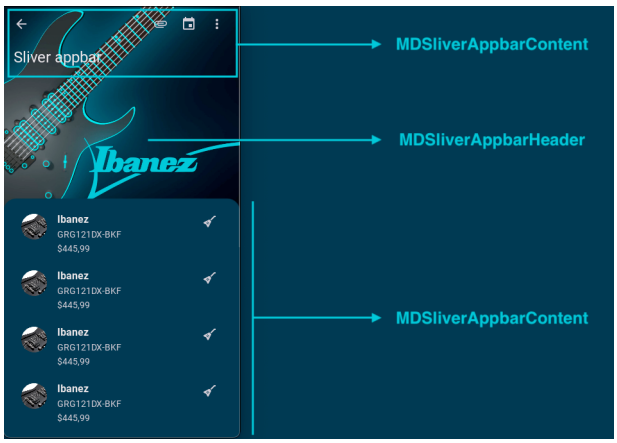
MDSliverAppbar is a Material Design widget in KivyMD which gives scrollable or collapsible MDTopAppBar

Note: This widget is a modification of the silverappbar.py module.

Usage

```
MDScreen:

    MDSliverAppbar:

        MDTopAppBar:
            [...]

        MDSliverAppbarHeader:

            # Custom content.
            [...]

        # Custom list.
        MDSliverAppbarContent:
```

Anatomy



Example

```
from kivy.lang.builder import Builder

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

KV = '''
<GuitarItem>
    theme_bg_color: "Custom"
    md_bg_color: "2d4a50"

    MDListItemLeadingAvatar
        source: "avatar.png"

    MDListItemHeadlineText:
```

(continues on next page)

(continued from previous page)

```
        text: "Ibanez"

    MDListItemSupportingText:
        text: "GRG121DX-BKF"

    MDListItemTertiaryText:
        text: "$445,99"

    MDListItemTrailingIcon:
        icon: "guitar-electric"


MDScreen:

    MDSliverAppbar:
        background_color: "2d4a50"
        hide_appbar: True

        MDTopAppBar:
            type: "medium"

            MDTopAppBarLeadingButtonContainer:

                MDActionTopAppBarButton:
                    icon: "arrow-left"

            MDTopAppBarTitle:
                text: "Sliver toolbar"

            MDTopAppBarTrailingButtonContainer:

                MDActionTopAppBarButton:
                    icon: "attachment"

                MDActionTopAppBarButton:
                    icon: "calendar"

                MDActionTopAppBarButton:
                    icon: "dots-vertical"

        MDSliverAppbarHeader:

            FitImage:
                source: "bg.jpg"

        MDSliverAppbarContent:
            id: content
            orientation: "vertical"
            padding: "12dp"
            theme_bg_color: "Custom"
            md_bg_color: "2d4a50"
'''
```

(continues on next page)

(continued from previous page)

```
class GuitarItem(MDListItem):
    ...


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

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


Example().run()
```

API break

1.2.0 version

```
#:import SliverToolbar __main__.SliverToolbar

Root:

    MDSliverAppbar:
        [...]

        MDSliverAppbarHeader:

            [...]

        MDSliverAppbarContent:
            [...]
```

```
class SliverToolbar(MDTopAppBar):
    [...]
```

2.0.0 version

```
Root:

    MDSliverAppbar:
        [...]

        MDTopAppBar:
            [...]
```

(continues on next page)

(continued from previous page)

```
        MDSliverAppbarHeader:

            [...]

        MDSliverAppbarContent:
            [...]
```

API - kivymd.uix.sliverappbar.sliverappbar

class kivymd.uix.sliverappbar.sliverappbar.**MDSliverAppbarContent**(**args*, ***kwargs*)

Implements a box for a scrollable list of custom items.

For more information, see in the *MDBoxLayout* class documentation.

class kivymd.uix.sliverappbar.sliverappbar.**MDSliverAppbarHeader**(***kwargs*)

Sliver app bar header class.

For more information, see in the BoxLayout class documentation.

class kivymd.uix.sliverappbar.sliverappbar.**MDSliverAppbar**(***kwargs*)

Sliver appbar class.

For more information, see in the *ThemableBehavior* and BoxLayout classes documentation.

Events

on_scroll_content
Fired when the list of custom content is being scrolled.

background_color

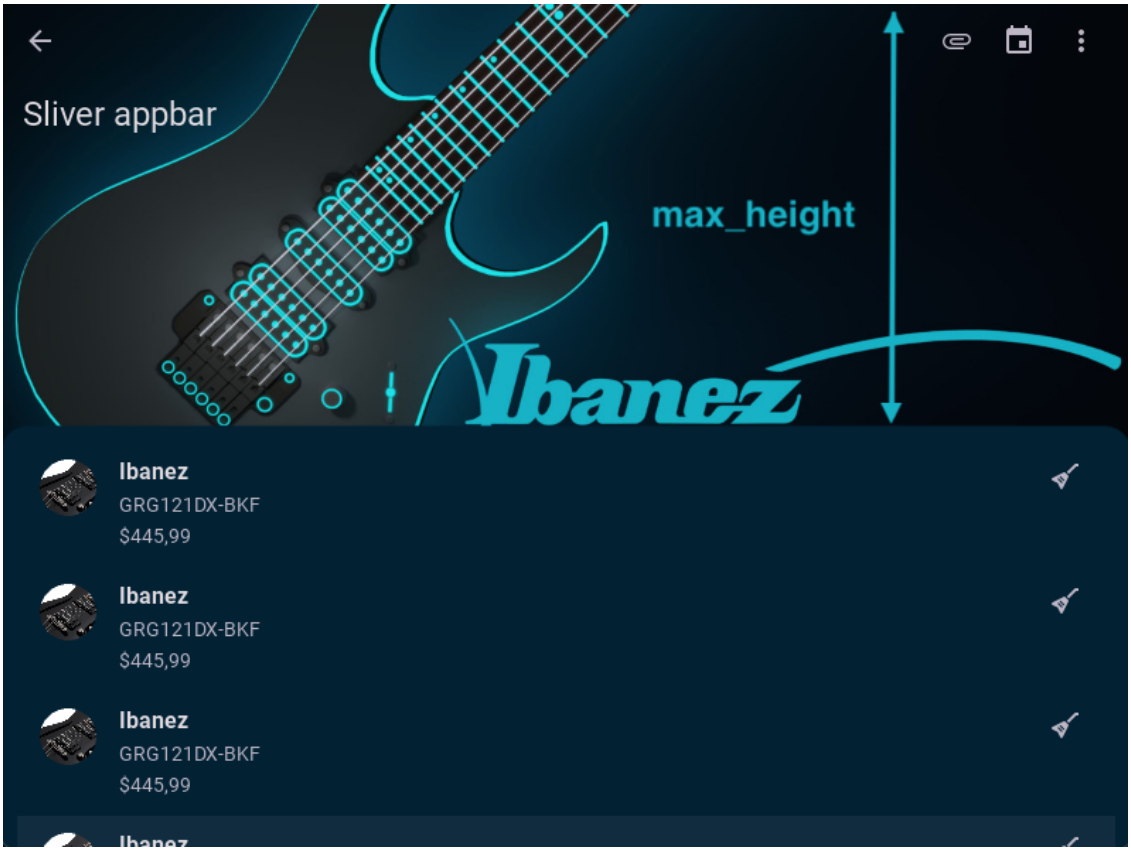
Background color of appbar in (r, g, b, a) or string format.

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_appbar

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

Changed in version 2.0.0: Rename *hide_toolbar* to *hide_appbar* attribute.

```
MDSliverAppbar:
    hide_appbar: False
```

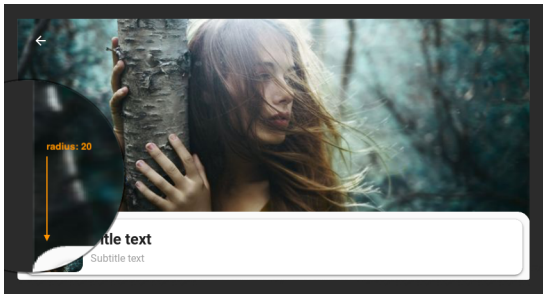
```
MDSliverAppbar:
    hide_appbar: True
```

hide_appbar is an BooleanProperty and defaults to *None*.

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_hide_appbar(*instance*, *value*) → None

Fired when the *hide_appbar* value changes.

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

Fired when the list of custom content is being scrolled.

Parameters

- **instance** – *MDSliverAppbar*
- **value** – see scroll_y
- **direction** – scroll direction: 'up/down'

on_background_color(*instance*, *color*) → None

Fired when the *background_color* value changes.

on_vbar() → None

add_widget(*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)
```

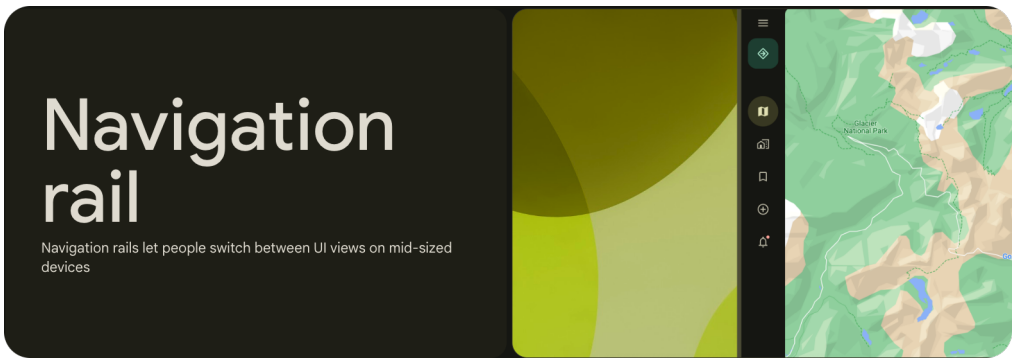
on__appbar(*instance*, *value*)

2.3.46 NavigationRail

New in version 1.0.0.

See also:

Material Design spec, Navigation rail



Navigation rails let people switch between UI views on mid-sized devices.

- Can contain 3-7 destinations plus an optional FAB
- Always put the rail in the same place, even on different screens of an app

Example

Declarative KV style

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

from kivymd.app import MDApp
from kivymd.uix.navigationrail import MDNavigationRailItem

KV = '''
<CommonNavigationRailItem>
```

(continues on next page)

(continued from previous page)

```
    MDNavigationRailItemIcon:
        icon: root.icon

    MDNavigationRailItemLabel:
        text: root.text


MDBoxLayout:

    MDNavigationRail:
        type: "selected"

        MDNavigationRailMenuButton:
            icon: "menu"

        MDNavigationRailFabButton:
            icon: "home"

        CommonNavigationRailItem:
            icon: "folder-outline"
            text: "Files"

        CommonNavigationRailItem:
            icon: "bookmark-outline"
            text: "Bookmark"

        CommonNavigationRailItem:
            icon: "library-outline"
            text: "Library"

    MDScreen:
        md_bg_color: self.theme_cls.secondaryContainerColor
'''


class CommonNavigationRailItem(MDNavigationRailItem):
    text = StringProperty()
    icon = StringProperty()


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


Example().run()
```

Declarative python style

```
from kivy.clock import Clock
from kivy.properties import StringProperty
```

(continues on next page)

(continued from previous page)

```
from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.navigationrail import (
    MDNavigationRailItem,
    MDNavigationRail,
    MDNavigationRailMenuButton,
    MDNavigationRailFabButton,
    MDNavigationRailItemIcon,
    MDNavigationRailItemLabel,
)
from kivymd.uix.screen import MDScreen


class CommonNavigationRailItem(MDNavigationRailItem):
    text = StringProperty()
    icon = StringProperty()

    def on_icon(self, instance, value):
        def on_icon(*ars):
            self.add_widget(MDNavigationRailItemIcon(icon=value))
        Clock.schedule_once(on_icon)

    def on_text(self, instance, value):
        def on_text(*ars):
            self.add_widget(MDNavigationRailItemLabel(text=value))
        Clock.schedule_once(on_text)


class Example(MDApp):
    def build(self):
        return MDBoxLayout(
            MDNavigationRail(
                MDNavigationRailMenuButton(
                    icon="menu",
                ),
                MDNavigationRailFabButton(
                    icon="home",
                ),
                CommonNavigationRailItem(
                    icon="bookmark-outline",
                    text="Files",
                ),
                CommonNavigationRailItem(
                    icon="folder-outline",
                    text="Bookmark",
                ),
                CommonNavigationRailItem(
                    icon="library-outline",
                    text="Library",
                ),
                type="selected",
            ),
```

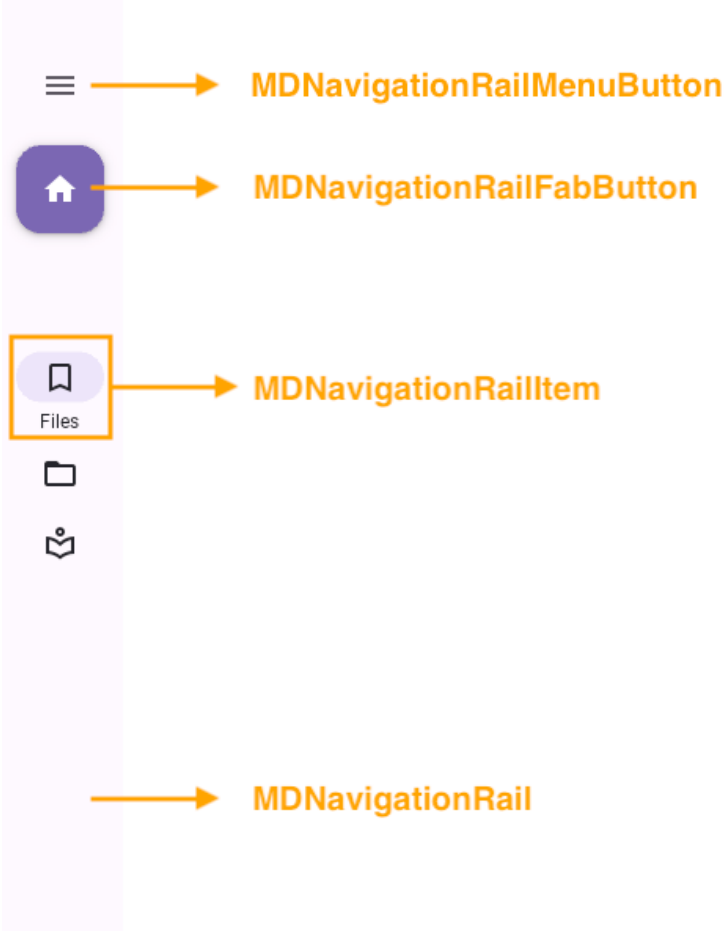
(continues on next page)

(continued from previous page)

```
            MDScreen(
                md_bg_color=self.theme_cls.secondaryContainerColor,
            ),
        )


Example().run()
```

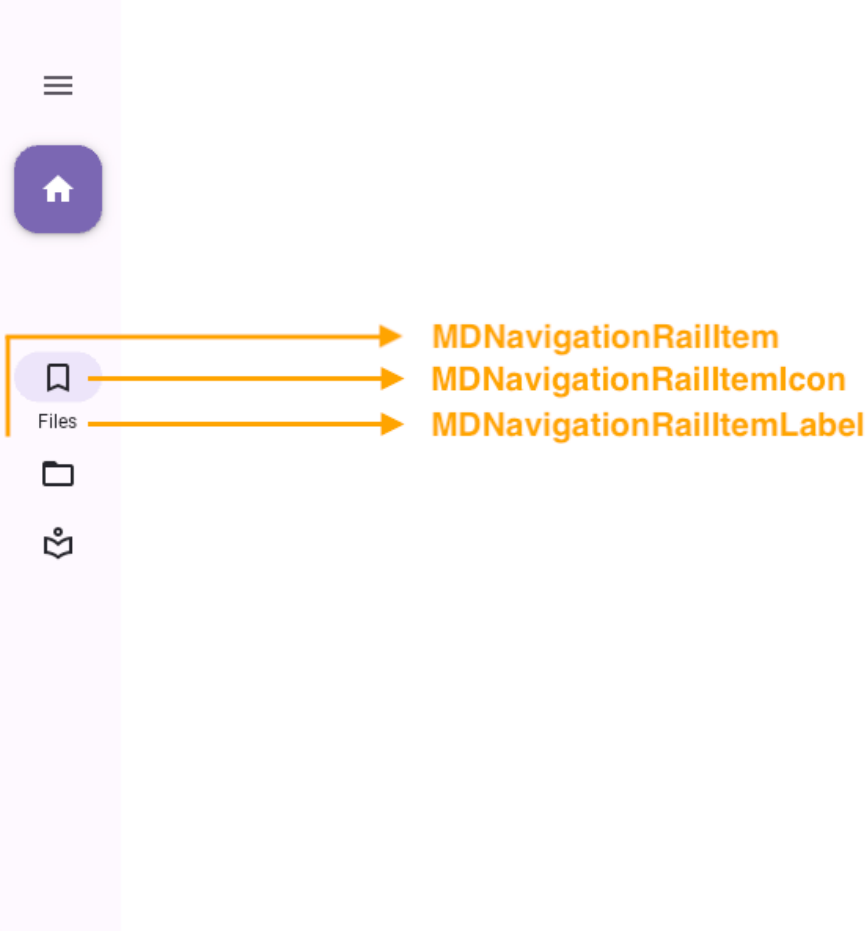
Anatomy

```
MDNavigationRail:

    # Optional.
    MDNavigationRailMenuButton:
        icon: "menu"

    # Optional.
    MDNavigationRailFabButton:
        icon: "home"

    MDNavigationRailItem

        MDNavigationRailItemIcon:
            icon: icon

        MDNavigationRailItemLabel:
            text: text

    [...]
```



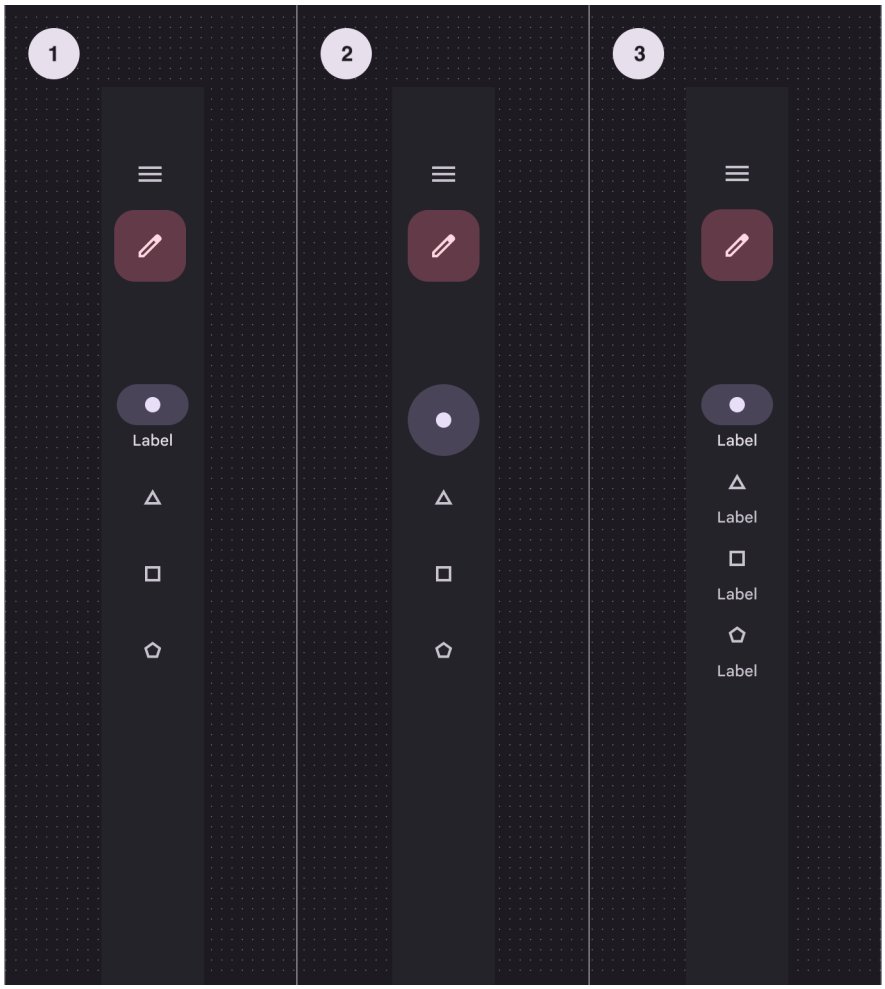
Anatomy item

```
MDNavigationRailItem

    MDNavigationRailItemIcon:
        icon: icon

    MDNavigationRailItemLabel:
        text: text
```



Configurations

Rail types



1. Selected
2. Unselected
3. Labeled

Selected

```
MDNavigationRail:
    type: "selected"  # default
```

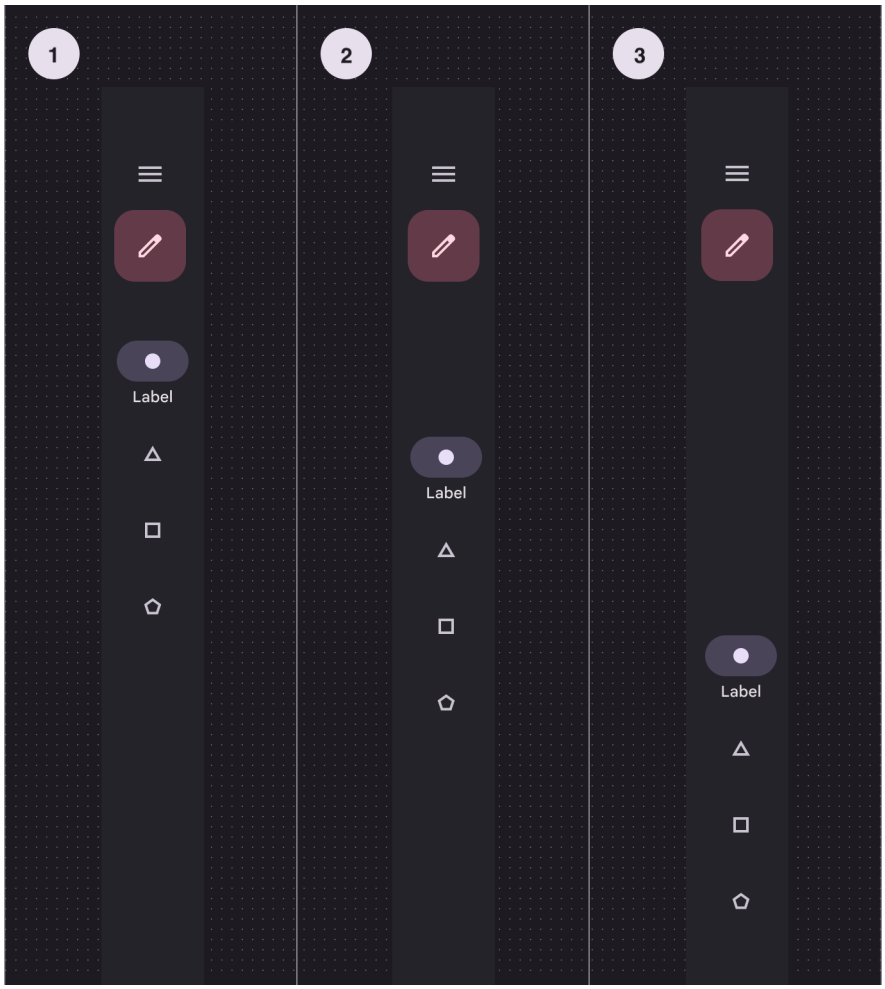
Unselected

```
MDNavigationRail:
    type: "unselected"
```

Labeled

```
MDNavigationRail:
    type: "labeled"
```

Rail anchored



1. Top
2. Center
3. Bottom

Top

```
MDNavigationRail:
    anchor: "top"
```

Center

```
MDNavigationRail:
    anchor: "center"  # default
```

Bottom

```
MDNavigationRail:
    anchor: "bottom"
```

API break

1.2.0 version

```
MDNavigationRail:

    MDNavigationRailMenuButton:
        icon: "menu"

    MDNavigationRailFabButton:
        icon: "home"

    MDNavigationRailItem:
        icon: icon
        text: text

    [...]
```

2.2.0 version

```
MDNavigationRail:

    MDNavigationRailMenuButton:
        icon: "menu"

    MDNavigationRailFabButton:
        icon: "home"

    MDNavigationRailItem
```

(continues on next page)

(continued from previous page)

```
        MDNavigationRailItemIcon:
            icon: icon

        MDNavigationRailItemLabel:
            text: text

    [...]
```

API - kivymd.uix.navigationrail.navigationrail

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

Implements a floating action button (FAB).

For more information, see in the *MDFabButton* class documentation.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the switch when the widget is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

class kivymd.uix.navigationrail.navigationrail.**MDNavigationRailMenuButton**(***kwargs*)

Implements a menu button.

For more information, see in the *MDIconButton* class documentation.

md_bg_color_disabled

The background color in (r, g, b, a) or string format of the switch when the widget is disabled.

md_bg_color_disabled is a ColorProperty and defaults to *None*.

class kivymd.uix.navigationrail.navigationrail.**MDNavigationRailItemIcon**(**args*, ***kwargs*)

Implements an icon for the *MDNavigationRailItem* class.

For more information, see in the *RectangularRippleBehavior* and *MDIcon* classes documentation.

Changed in version 2.0.0.

active_indicator_color

Background color of the active indicator in (r, g, b, a) or string format.

active_indicator_color is an ColorProperty and defaults to *None*.

anim_complete(**args*)

Fired when the “fade_out” animation complete.

lay_canvas_instructions() → None

Adds graphic instructions to the canvas to implement ripple animation.

class kivymd.uix.navigationrail.navigationrail.**MDNavigationRailItemLabel**(**args*, ***kwargs*)

Implements an label for the *MDNavigationRailItem* class.

For more information, see in the *ScaleBehavior* and *MDLabel* classes documentation.

Changed in version 2.0.0.

scale_value_y

Y-axis value.

scale_value_y is an NumericProperty and defaults to *0*.

on__active(*instance*, *value*) → None

Fired when the _active value changes.

class kivymd.uix.navigationrail.navigationrail.**MDNavigationRailItem**(**args*, ***kwargs*)

Implements a menu item with an icon and text.

For more information, see in the *DeclarativeBehavior* and ButtonBehavior and *ThemableBehavior* and *FocusBehavior* BoxLayout classes documentation.

active

Is the element active.

active is an BooleanProperty and defaults to *False*.

radius

Item radius.

Changed in version 2.0.0.

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

on_active(*instance*, *value*) → None

Fired when the *active* value changes.

on_enter(**args*) → None

Fired when mouse enter the bbox of the widget.

on_leave(**args*) → None

Fired when the mouse goes outside the widget border.

add_widget(*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())
```

(continues on next page)

(continued from previous page)

```
>>> slider = Slider()
>>> root.add_widget(slider)
```

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

Navigation rail class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and RelativeLayout classes documentation.

radius

Rail radius.

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

anchor

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

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

type

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

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

fab_button: *MDNavigationRailFabButton*

menu_button: *MDNavigationRailFabButton*

on_size(**args*) → None

Fired when the application screen size changes.

get_items() → list

Returns a list of *MDNavigationRailItem* objects.

set_active_item(*item:* MDNavigationRailItem) → None

Sets the active menu list item.

add_widget(*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.

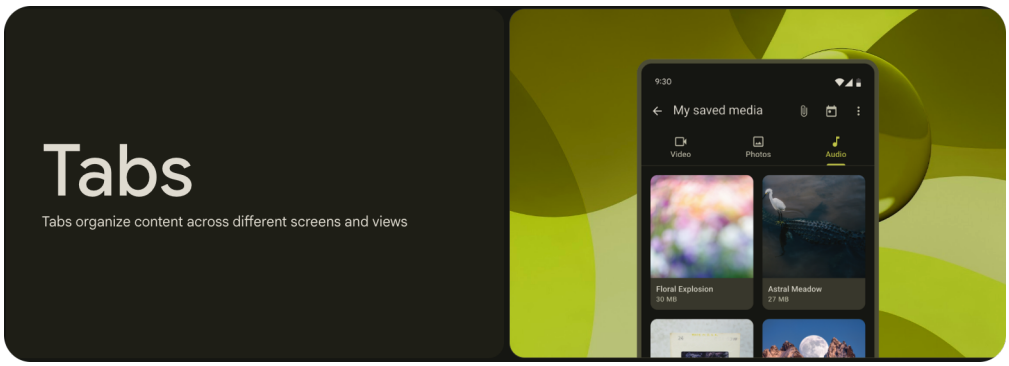
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.47 Tabs

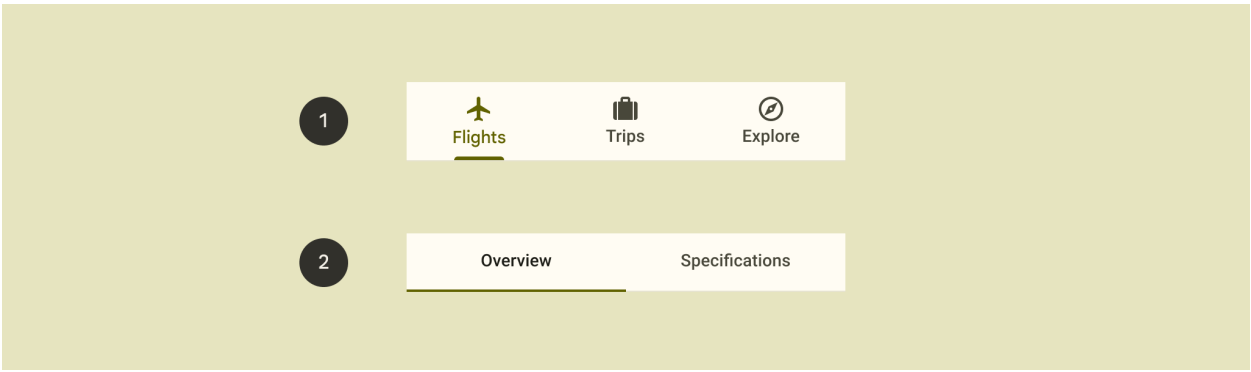
See also:

Material Design spec, Tabs

Tabs organize content across different screens, data sets, and other interactions.



- Use tabs to group content into helpful categories
- Two types: primary and secondary
- Tabs can horizontally scroll, so a UI can have as many tabs as needed
- Place tabs next to each other as peers



1. Primary tabs
2. Secondary tabs

Usage primary tabs

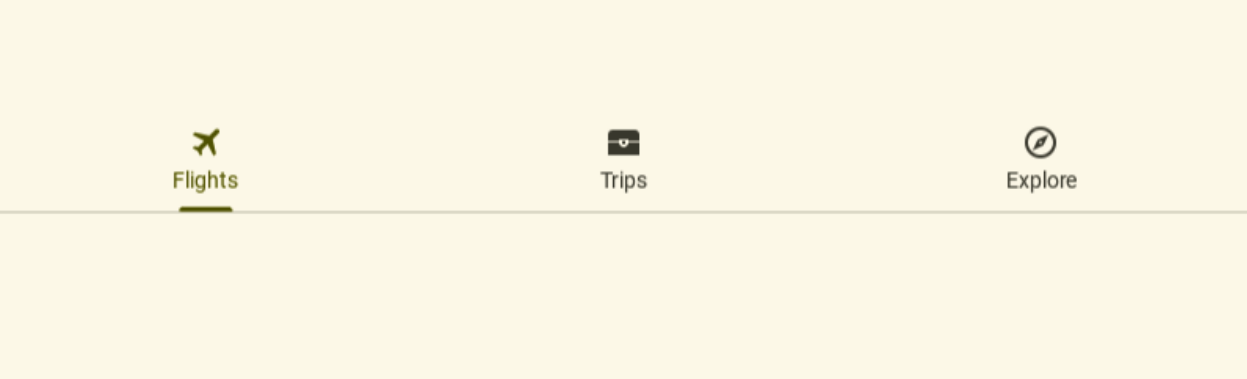
Primary tabs should be used when just one set of tabs are needed.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import (
    MDTabsItem,
    MDTabsItemIcon,
    MDTabsItemText,
    MDTabsBadge,
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTabsPrimary:
        id: tabs
        pos_hint: {"center_x": .5, "center_y": .5}

        MDDivider:
'''


class Example(MDApp):
    def on_start(self):
        for tab_icon, tab_name in {
            "airplane": "Flights",
            "treasure-chest": "Trips",
            "compass-outline": "Explore",
        }.items():
            if tab_icon == "treasure-chest":
                self.root.ids.tabs.add_widget(
                    MDTabsItem(
                        MDTabsItemIcon(
                            MDTabsBadge(
                                text="99",
                            ),
                            icon=tab_icon,
                        ),
                        MDTabsItemText(
                            text=tab_name,
                        ),
                    )
                )
            else:
                self.root.ids.tabs.add_widget(
                    MDTabsItem(
                        MDTabsItemIcon(
                            icon=tab_icon,
                        ),
```

(continues on next page)

(continued from previous page)

```
                        MDTabsItemText(
                            text=tab_name,
                        ),
                    )
                )
            self.root.ids.tabs.switch_tab(icon="airplane")

    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)


Example().run()
```



Anatomy primary tabs

Usage secondary tabs

Secondary tabs are necessary when a screen requires more than one level of tabs. These tabs use a simpler style of indicator, but their function is identical to primary tabs.

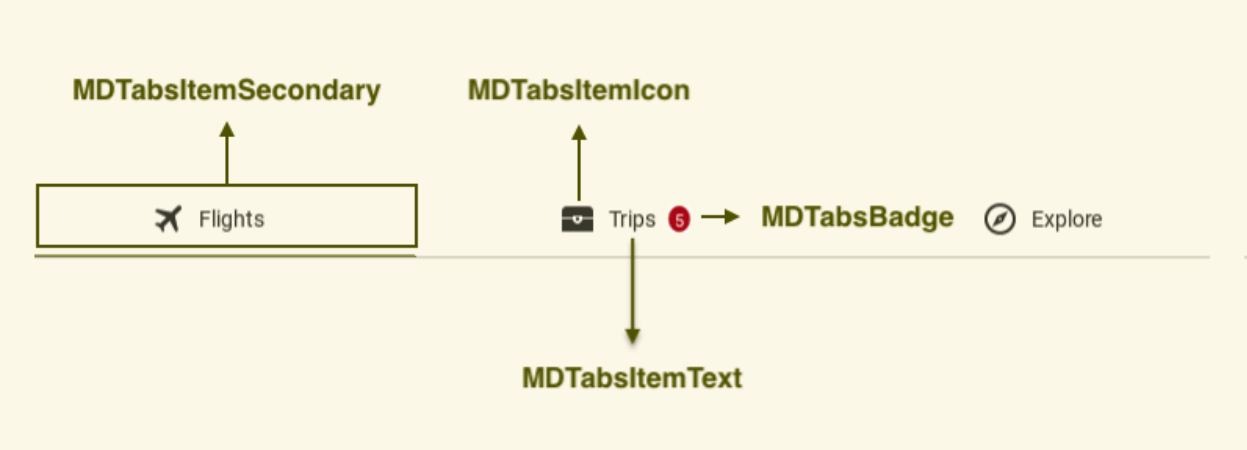
```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import (
    MDTabsItemIcon,
    MDTabsItemText,
    MDTabsBadge, MDTabsItemSecondary,
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTabsSecondary:
        id: tabs
        pos_hint: {"center_x": .5, "center_y": .5}

        MDDivider:
'''


class Example(MDApp):
    def on_start(self):
        for tab_icon, tab_name in {
            "airplane": "Flights",
            "treasure-chest": "Trips",
            "compass-outline": "Explore",
        }.items():
            if tab_icon == "treasure-chest":
                self.root.ids.tabs.add_widget(
                    MDTabsItemSecondary(
                        MDTabsItemIcon(
                            icon=tab_icon,
                        ),
                        MDTabsItemText(
                            text=tab_name,
                        ),
                        MDTabsBadge(
                            text="5",
                        ),
                    )
                )
            else:
                self.root.ids.tabs.add_widget(
                    MDTabsItemSecondary(
                        MDTabsItemIcon(
                            icon=tab_icon,
                        ),
```

(continues on next page)

(continued from previous page)

```
                        MDTabsItemText(
                            text=tab_name,
                        ),
                    )
                )
        self.root.ids.tabs.switch_tab(icon="airplane")

    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)


Example().run()
```

Anatomy secondary tabs



Related content

Use tabs to group related content, not sequential content.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.label import MDLabel
from kivymd.uix.tab import (
    MDTabsItemIcon,
    MDTabsItemText,
    MDTabsItem,
)

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor
```

(continues on next page)

(continued from previous page)

```
    MDTabsPrimary:
        id: tabs
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint_x: .6

        MDDivider:

        MDTabsCarousel:
            id: related_content_container
            size_hint_y: None
            height: dp(320)
'''


class Example(MDApp):
    def on_start(self):
        for tab_icon, tab_name in {
            "airplane": "Flights",
            "treasure-chest": "Trips",
            "compass-outline": "Explore",
        }.items():
            self.root.ids.tabs.add_widget(
                MDTabsItem(
                    MDTabsItemIcon(
                        icon=tab_icon,
                    ),
                    MDTabsItemText(
                        text=tab_name,
                    ),
                )
            )
            self.root.ids.related_content_container.add_widget(
                MDLabel(
                    text=tab_name,
                    halign="center",
                )
            )
            self.root.ids.tabs.switch_tab(icon="airplane")

    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)


Example().run()
```

Behaviors

Scrollable tabs

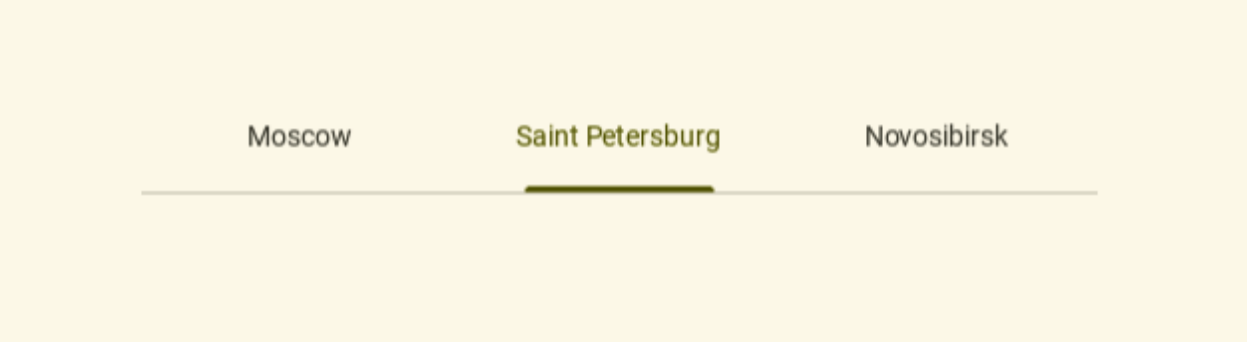
When a set of tabs cannot fit on screen, use scrollable tabs. Scrollable tabs can use longer text labels and a larger number of tabs. They are best used for browsing on touch interfaces.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsItemText, MDTabsItem

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTabsPrimary:
        id: tabs
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint_x: .6
        allow_stretch: False
        label_only: True

        MDDivider:
'''


class Example(MDApp):
    def on_start(self):
        for tab_name in [
            "Moscow",
            "Saint Petersburg",
            "Novosibirsk",
            "Yekaterinburg",
            "Kazan",
            "Nizhny Novgorod",
            "Chelyabinsk",
        ]:
            self.root.ids.tabs.add_widget(
                MDTabsItem(
                    MDTabsItemText(
                        text=tab_name,
                    ),
                )
            )
        self.root.ids.tabs.switch_tab(text="Moscow")

    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)


Example().run()
```

Fixed tabs

Fixed tabs display all tabs in a set simultaneously. They are best for switching between related content quickly, such as between transportation methods in a map. To navigate between fixed tabs, tap an individual tab, or swipe left or right in the content area.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.tab import MDTabsItemText, MDTabsItem

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTabsPrimary:
        id: tabs
        pos_hint: {"center_x": .5, "center_y": .5}
        size_hint_x: .6
        allow_stretch: True
        label_only: True

        MDDivider:
'''


class Example(MDApp):
    def on_start(self):
        for tab_name in [
            "Moscow", "Saint Petersburg", "Novosibirsk"
        ]:
            self.root.ids.tabs.add_widget(
                MDTabsItem(
                    MDTabsItemText(
                        text=tab_name,
                    ),
                )
            )
        self.root.ids.tabs.switch_tab(text="Moscow")

    def build(self):
        self.theme_cls.primary_palette = "Olive"
        return Builder.load_string(KV)


Example().run()
```



Tap a tab

Navigate to a tab by tapping on it.

Swipe within the content area

To navigate between tabs, users can swipe left or right within the content area.

Switching tab

You can switch tabs by icon name, by tab name, and by tab objects:

```
instance_tabs.switch_tab(icon="airplane")
```

```
instance_tabs.switch_tab(text="Airplane")
```

```
instance_tabs.switch_tab(
    instance=instance_tabs_item  # MDTabsItem
)
```

API break

1.2.0 version

```
from kivy.lang import Builder

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

KV = '''
MDBoxLayout:
```

(continues on next page)

(continued from previous page)

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

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


Example().run()
```

2.0.0 version

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.icon_definitions import md_icons
from kivymd.uix.label import MDIcon
from kivymd.uix.tab import MDTabsItem, MDTabsItemIcon
from kivymd.uix.tab.tab import MDTabsItemText

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDTabsPrimary:
```

(continues on next page)

(continued from previous page)

```
        id: tabs
        allow_stretch: False
        pos_hint: {"center_x": .5, "center_y": .5}

        MDDivider:

        MDTabsCarousel:
            id: related_content
            size_hint_y: None
            height: root.height - tabs.ids.tab_scroll.height
'''


class Example(MDApp):
    def on_start(self):
        for name_tab in list(md_icons.keys())[15:30]:
            self.root.ids.tabs.add_widget(
                MDTabsItem(
                    MDTabsItemIcon(
                        icon=name_tab,
                    ),
                    MDTabsItemText(
                        text=name_tab,
                    ),
                )
            )
            self.root.ids.related_content.add_widget(
                MDIcon(
                    icon=name_tab,
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                )
            )
            self.root.ids.tabs.switch_tab(icon="airplane")

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

API - kivymd.uix.tab.tab

class kivymd.uix.tab.tab.**MDTabsBadge**(**args*, ***kwargs*)

Implements an badge for secondary tabs.

New in version 2.0.0.

For more information, see in the *MDBadge* class documentation.

class kivymd.uix.tab.tab.**MDTabsCarousel**(***kwargs*)

Implements a carousel for user-generated content.

For more information, see in the Carousel class documentation.

lock_swiping

If True - disable switching tabs by swipe.

lock_swiping is an BooleanProperty and defaults to *False*.

on_touch_move(*touch*) → str | bool | None

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

See on_touch_down() for more information.

class kivymd.uix.tab.tab.**MDTabsItemText**(**args*, ***kwargs*)

Implements an label for the *MDTabsItem* class.

For more information, see in the *MDLabel* class documentation.

New in version 2.0.0.

class kivymd.uix.tab.tab.**MDTabsItemIcon**(**args*, ***kwargs*)

Implements an icon for the *MDTabsItem* class.

For more information, see in the *MDIcon* class documentation.

New in version 2.0.0.

class kivymd.uix.tab.tab.**MDTabsItem**(**args*, ***kwargs*)

Implements a item with an icon and text for *MDTabsPrimary* class.

New in version 2.0.0.

For more information, see in the MDTabsItemBase and BoxLayout classes documentation.

add_widget(*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)
```

class kivymd.uix.tab.tab.**MDTabsPrimary**(**args*, ***kwargs*)

Tabs primary class.

Changed in version 2.0.0: Rename from *MDTabs* to *MDTabsPrimary* class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and BoxLayout classes documentation.

Events

on_tab_switch
Fired when switching tabs.

md_bg_color

The background color of the widget.

md_bg_color is an ColorProperty and defaults to *None*.

label_only

Tabs with a label only or with an icon and a label.

New in version 2.0.0.

label_only is an BooleanProperty and defaults to *False*.

allow_stretch

Whether to stretch tabs to the width of the panel.

allow_stretch is an BooleanProperty and defaults to *True*.

lock_swiping

If True - disable switching tabs by swipe.

lock_swiping is an BooleanProperty and defaults to *False*.

anim_duration

Duration of the slide animation.

anim_duration is an NumericProperty and defaults to *0.2*.

indicator_anim

Tab indicator animation. If you want use animation set it to True.

Changed in version 2.0.0: Rename from *tab_indicator_anim* to *indicator_anim* attribute.

indicator_anim is an BooleanProperty and defaults to *True*.

indicator_radius

Radius of the tab indicator.

New in version 2.0.0.

indicator_radius is an VariableListProperty and defaults to *[dp(2), dp(2), 0, 0]*.

indicator_height

Height of the tab indicator.

Changed in version 2.0.0: Rename from *tab_indicator_height* to *indicator_height* attribute.

indicator_height is an NumericProperty and defaults to *'4dp'*.

indicator_duration

The duration of the animation of the indicator movement when switching tabs.

New in version 2.0.0.

indicator_duration is an NumericProperty and defaults to *0.5*.

indicator_transition

The transition name of animation of the indicator movement when switching tabs.

New in version 2.0.0.

indicator_transition is an StringProperty and defaults to `'out_expo'.

last_scroll_x

Is the carousel reference of the next tab/slide. When you go from *'Tab A'* to *'Tab B'*, *'Tab B'* will be the target tab/slide of the carousel.

last_scroll_x is an AliasProperty.

target

It is the carousel reference of the next tab / slide. When you go from *'Tab A'* to *'Tab B'*, *'Tab B'* will be the target tab / slide of the carousel.

target is an ObjectProperty and default to *None*.

indicator

It is the SmoothRoundedRectangle instruction reference of the tab indicator.

indicator is an AliasProperty.

get_last_scroll_x()

get_rect_instruction()

add_widget(*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())
```

(continues on next page)

(continued from previous page)

```
>>> slider = Slider()
>>> root.add_widget(slider)
```

do_autoscroll_tabs(*instance:* MDTabsItem, *value:* *float*) → None

Automatically scrolls the list of tabs when swiping the carousel slide (related content).

Changed in version 2.0.0: Rename from *tab_bar_autoscroll* to *do_autoscroll_tabs* method.

android_animation(*instance:* MDTabsCarousel, *offset:* *float*) → None

Fired when swiping a carousel slide (related content).

update_indicator(*x:* *float* = *0.0*, *w:* *float* = *0.0*, *instance:* MDTabsItem = *None*) → None

Update position and size of the indicator.

switch_tab(*instance:* MDTabsItem = *None*, *text:* *str* = *''*, *icon:* *str* = *''*) → None

Switches tabs by tab object/tab text/tab icon name.

set_active_item(*item:* MDTabsItem) → None

Sets the active tab item.

get_tabs_list() → list

Returns a list of *MDTabsItem* objects.

Changed in version 2.0.0: Rename from *get_tab_list* to *get_tabs_list* method.

get_slides_list() → list

Returns a list of user tab objects.

Changed in version 2.0.0: Rename from *get_slides* to *get_slides_list* method.

get_current_tab() → *MDTabsItem*

Returns current tab object.

New in version 1.0.0.

get_current_related_content() → kivy.uix.widget.Widget

Returns the carousel slide object (related content).

New in version 2.0.0.

on_tab_switch(**args*) → None

This event is launched every time the current tab is changed.

on_slide_progress(**args*) → None

This event is deployed every available frame while the tab is scrolling.

on_carousel_index(*instance:* MDTabsCarousel, *value:* *int*) → None

Fired when the Tab index have changed. This event is deployed by the builtin carousel of the class.

on_size(*instance*, *size*) → None

Fired when the application screen size changes.

class kivymd.uix.tab.tab.**MDTabsItemSecondary**(**args*, ***kwargs*)

Implements a item with an icon and text for *MDTabsSecondary* class.

New in version 2.0.0.

For more information, see in the MDTabsItemBase and AnchorLayout classes documentation.

add_widget(*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)
```

class kivymd.uix.tab.tab.**MDTabsSecondary**(**args*, ***kwargs*)

Tabs secondary class.

New in version 2.0.0.

For more information, see in the *MDTabsPrimary* class documentation.

indicator_radius

Radius of the tab indicator.

indicator_radius is an VariableListProperty and defaults to *[0, 0, 0, 0]*.

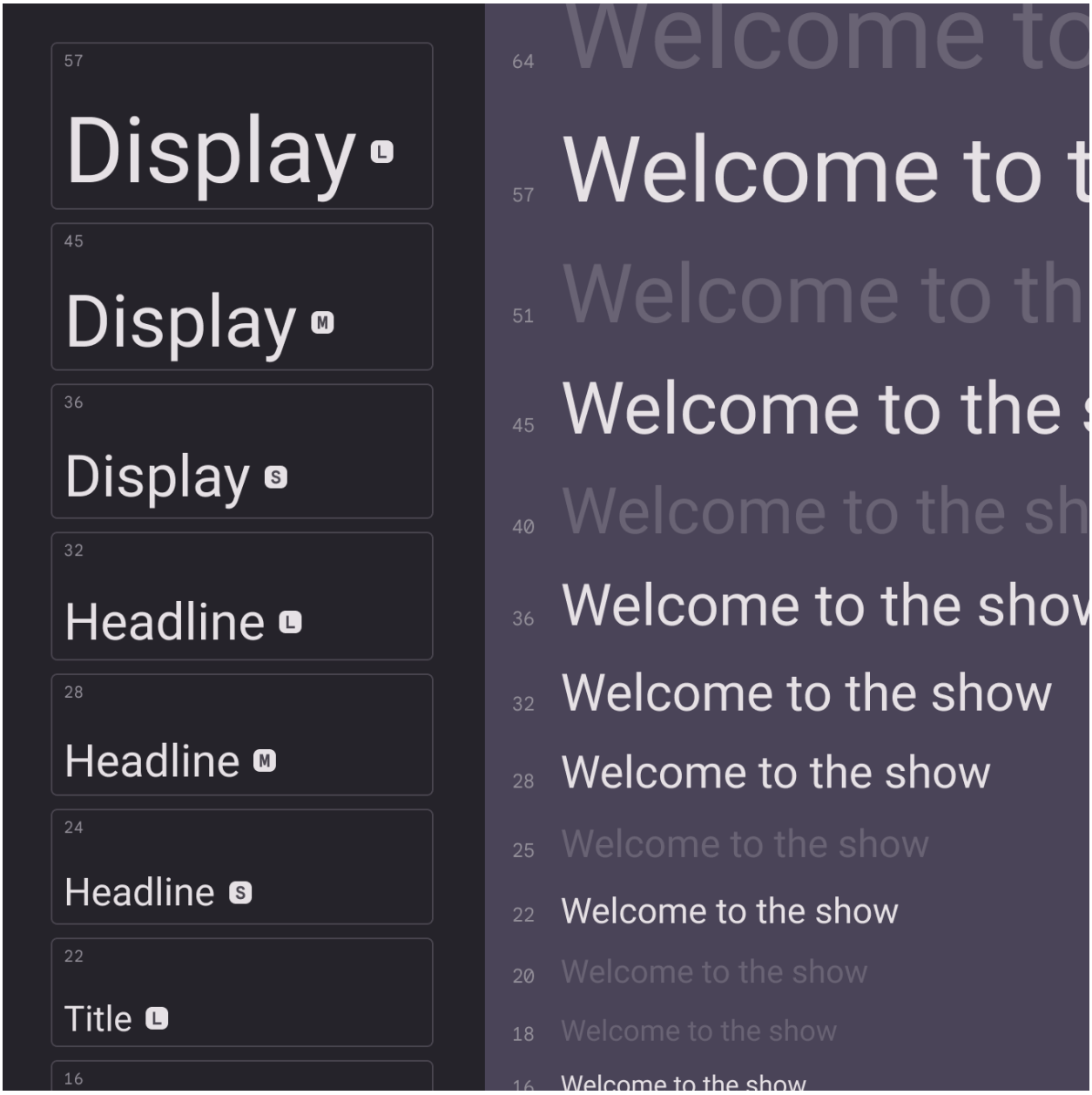
indicator_height

Height of the tab indicator.

indicator_height is an NumericProperty and defaults to *'2dp'*.

2.3.48 Label

The *MDLabel* widget is for rendering text.



- *MDLabel*
- *MDIcon*

MDLabel

Example

Declarative KV style

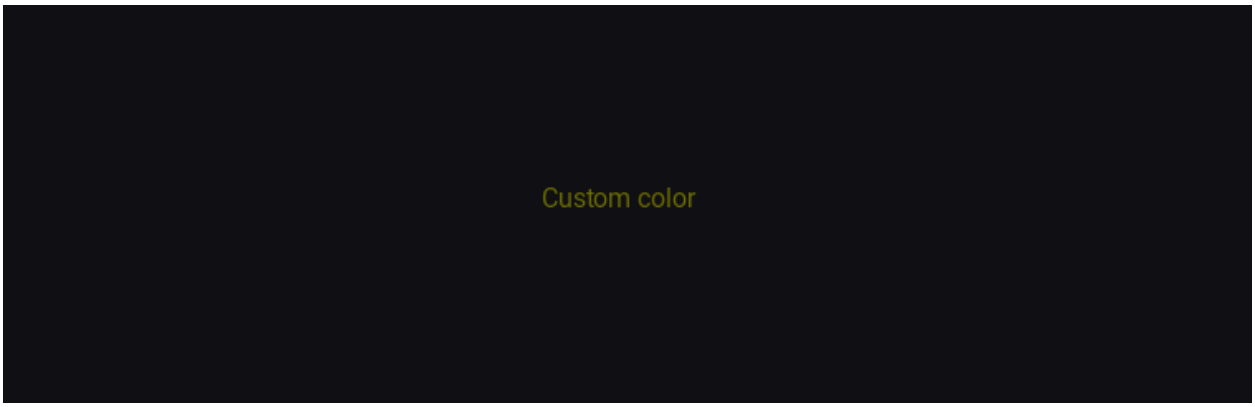
```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

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


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


Example().run()
```

Declarative Python style

```
from kivy.lang import Builder

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


class Test(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        return (
            MDScreen(
                MDLabel(
                    text="MDLabel",
                    halign="center",
                ),
                md_bg_color=self.theme_cls.backgroundColor,
            )
        )


Test().run()
```

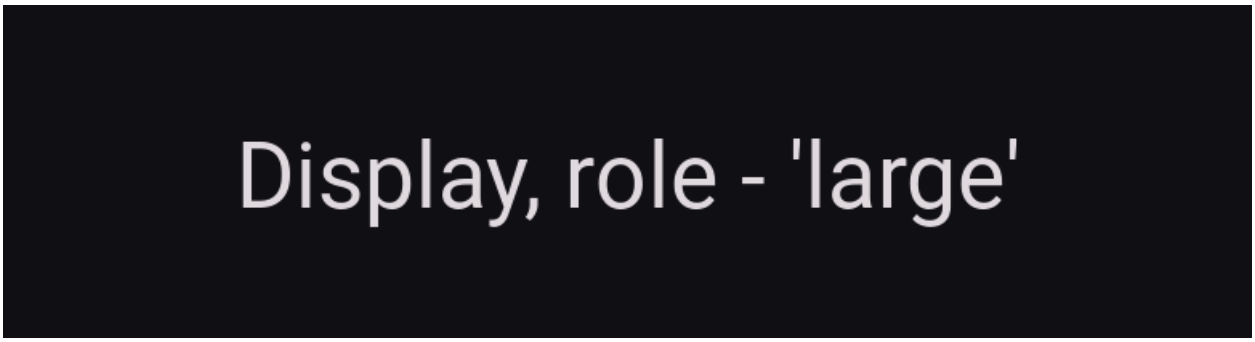
To use a custom color for *MDLabel*, use a theme *'Custom'*. After that, you can specify the desired color in the `text_color` parameter:

```
MDLabel:
    text: "Custom color"
    halin: "center"
    theme_text_color: "Custom"
    text_color: "red"
```

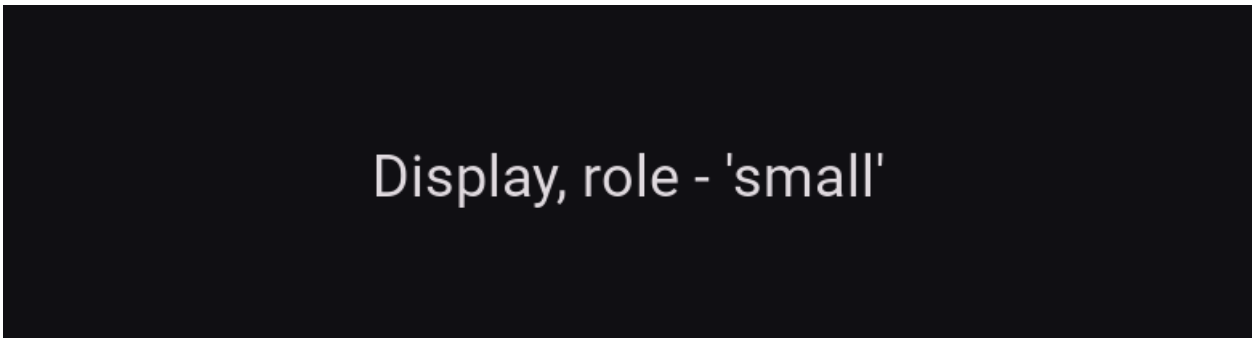


MDLabel provides standard font styles for labels. To do this, specify the name of the desired style in the *font_style* and *role* parameters:

```
MDLabel:
    text: "Display, role - 'large'"
    font_style: "Display"
```



```
MDLabel:
    text: "Display, role - 'small'"
    font_style: "Display"
    role: "small"
```



See also:

Material Design spec, Typography

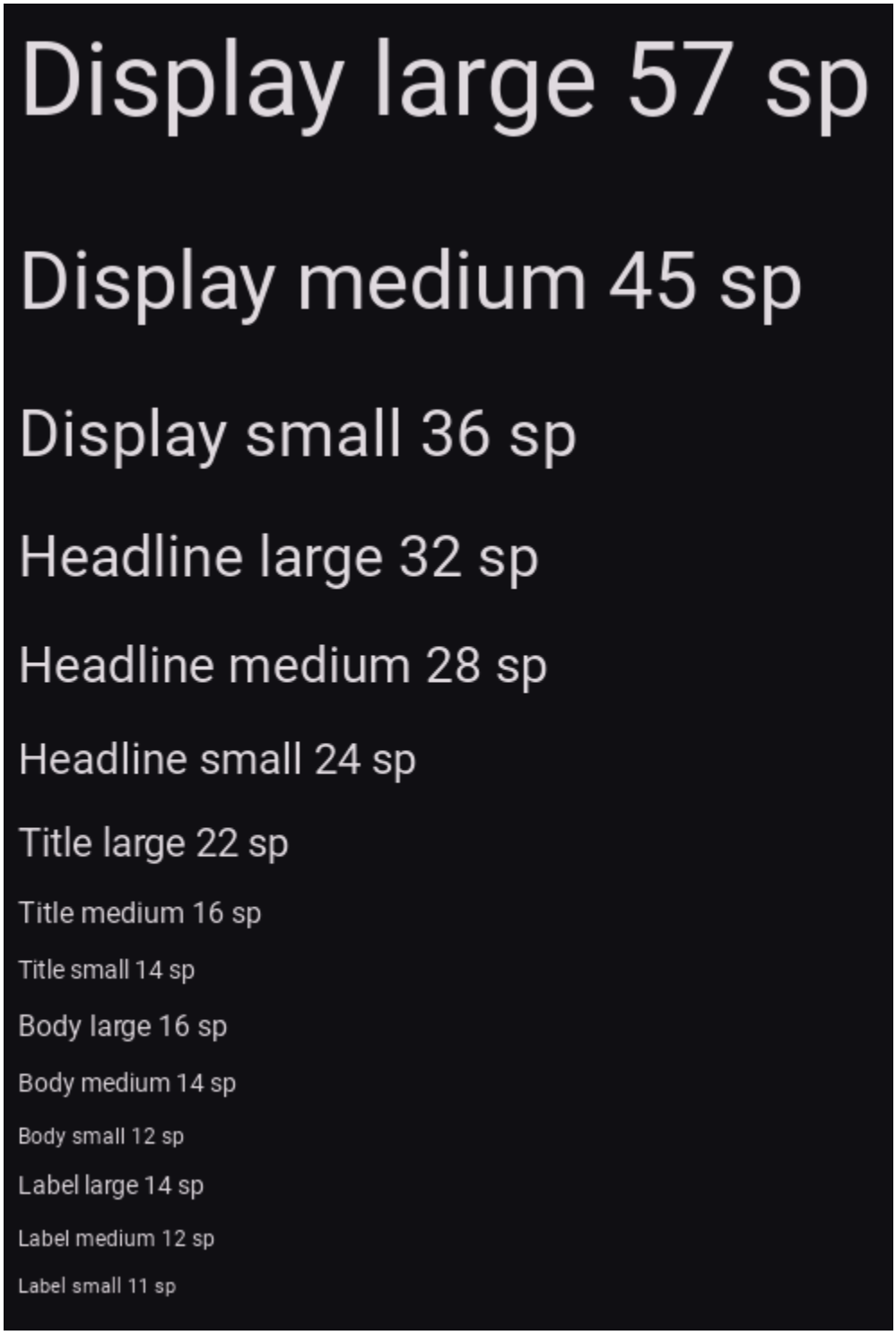
All styles

```
from kivy.lang import Builder

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

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

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

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


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

(continues on next page)

(continued from previous page)

```
    def on_start(self):
        for style in theme_font_styles:
            if style != "Icon":
                for role in theme_font_styles[style]:
                    font_size = int(theme_font_styles[style][role]["font-size"])
                    self.root.ids.rv.data.append(
                        {
                            "viewclass": "MDLabel",
                            "text": f"{style} {role} {font_size} sp",
                            "adaptive_height": "True",
                            "font_style": style,
                            "role": role,
                        }
                    )


Example().run()
```


Display large 57 sp
Display medium 45 sp
Display small 36 sp
Headline large 32 sp
Headline medium 28 sp
Headline small 24 sp
Title large 22 sp
Title medium 16 sp
Title small 14 sp
Body large 16 sp
Body medium 14 sp
Body small 12 sp
Label large 14 sp
Label medium 12 sp
Label small 11 sp

Highlighting and copying labels

You can highlight labels by double tap on the label:

Declarative KV style

```
from kivy.lang.builder import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDLabel:
        adaptive_size: True
        pos_hint: {"center_x": .5, "center_y": .5}
        text: "Do a double click on me"
        allow_selection: True
        padding: "4dp", "4dp"
'''


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


Example().run()
```

Declarative Python style

```
from kivy.clock import Clock

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


class Example(MDApp):
    def on_start(self):
        def on_start(dt):
            self.root.md_bg_color = self.theme_cls.backgroundColor

        Clock.schedule_once(on_start)

    def build(self):
        self.theme_cls.theme_style = "Dark"
        return (
            MDScreen(
                MDLabel(
                    adaptive_size=True,
```

(continues on next page)

(continued from previous page)

```
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                    text="Do a double click on me",
                    allow_selection=True,
                    padding=("4dp", "4dp"),
                ),
            )
        )


Example().run()
```

You can copy the label text by double clicking on it:

Declarative KV style

```
from kivy.lang.builder import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDLabel:
        adaptive_size: True
        pos_hint: {"center_x": .5, "center_y": .5}
        text: "MDLabel"
        padding: "4dp", "4dp"
        allow_selection: True
        allow_copy: True
        on_copy: print("The text is copied to the clipboard")
'''


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


Example().run()
```

Declarative Python style

```
from kivy.lang.builder import Builder

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

(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 (
            MDScreen(
                MDLabel(
                    id="label",
                    adaptive_size=True,
                    pos_hint={"center_x": .5, "center_y": .5},
                    text="MDLabel",
                    allow_selection=True,
                    allow_copy=True,
                    padding=("4dp", "4dp"),
                )
            )
        )

    def on_start(self):
        self.root.ids.label.bind(on_copy=self.on_copy)

    def on_copy(self, instance_label: MDLabel):
        print("The text is copied to the clipboard")


Example().run()
```

Example of copying/cutting labels using the context menu

```
from kivy.core.clipboard import Clipboard
from kivy.lang.builder import Builder
from kivy.metrics import dp

from kivymd.uix.snackbar import MDSnackbar, MDSnackbarText
from kivymd.app import MDApp
from kivymd.uix.label import MDLabel
from kivymd.uix.menu import MDDropdownMenu

KV = '''
MDBoxLayout:
    orientation: "vertical"
    spacing: "12dp"
    padding: "24dp"
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        id: box
        orientation: "vertical"
```

(continues on next page)

(continued from previous page)

```
        padding: "24dp"
        spacing: "12dp"
        adaptive_height: True

    MDTextField:
        max_height: "200dp"
        mode: "filled"
        multiline: True

    Widget:
'''

data = [
    "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
    "Sed blandit libero volutpat sed cras ornare arcu. Nisl vel pretium "
    "lectus quam id leo in. Tincidunt arcu non sodales neque sodales ut etiam.",
    "Elit scelerisque mauris pellentesque pulvinar pellentesque habitant. "
    "Nisl rhoncus mattis rhoncus urna neque. Orci nulla pellentesque "
    "dignissim enim. Ac auctor augue mauris augue neque gravida in fermentum. "
    "Lacus suspendisse faucibus interdum posuere."
]


def toast(text):
    MDSnackbar(
        MDSnackbarText(
            text=text,
        ),
        y=dp(24),
        pos_hint={"center_x": 0.5},
        size_hint_x=0.3,
    ).open()


class CopyLabel(MDLabel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.allow_selection = True
        self.adaptive_height = True


class Example(MDApp):
    context_menu = None

    def build(self):
        return Builder.load_string(KV)

    def on_start(self):
        for text in data:
            copy_label = CopyLabel(text=text)
            copy_label.bind(on_selection=self.open_context_menu)
            self.root.ids.box.add_widget(copy_label)
```

(continues on next page)

(continued from previous page)

```
    def click_item_context_menu(
        self, type_click: str, instance_label: CopyLabel
    ) -> None:
        Clipboard.copy(instance_label.text)

        if type_click == "copy":
            toast("Copied")
        elif type_click == "cut":
            self.root.ids.box.remove_widget(instance_label)
            toast("Cut")
        if self.context_menu:
            self.context_menu.dismiss()

    def open_context_menu(self, instance_label: CopyLabel) -> None:
        instance_label.text_color = "black"
        menu_items = [
            {
                "text": "Copy text",
                "on_release": lambda: self.click_item_context_menu(
                    "copy", instance_label
                ),
            },
            {
                "text": "Cut text",
                "on_release": lambda: self.click_item_context_menu(
                    "cut", instance_label
                ),
            },
        ]
        self.context_menu = MDDropdownMenu(
            caller=instance_label, items=menu_items, width_mult=3
        )
        self.context_menu.open()


Example().run()
```

MDIcon

You can use labels to display material design icons using the *MDIcon* class.

See also:

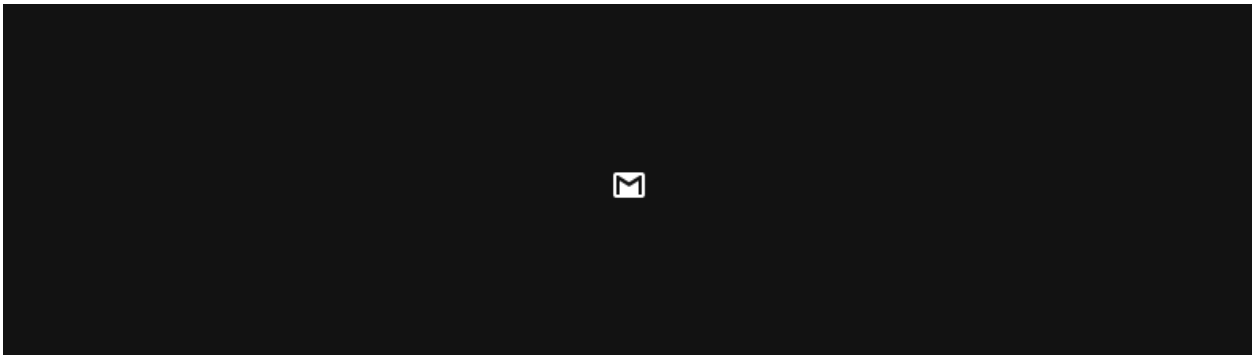
Material Design Icons

Material Design Icon Names

The *MDIcon* class is inherited from *MDLabel* and has the same parameters.

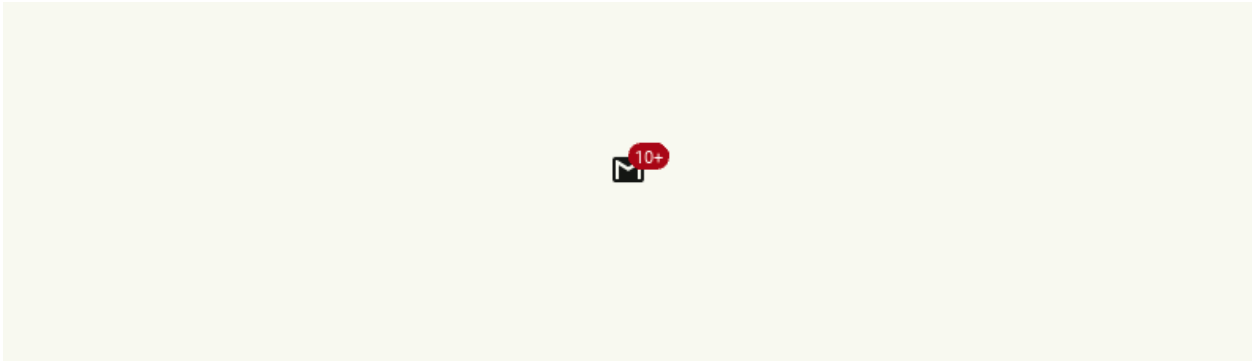
Warning: For the *MDIcon* class, you cannot use `text` and `font_style` options!

```
MDIcon:
    icon: "gmail"
```



MDIcon with badge icon

```
MDIcon:
    icon: "gmail"

    MDBadge:
        text: "10+"
```



MDIcon with a custom font icon

You can use custom fonts to display icons. Such as for example Material Symbols. You can find the necessary fonts in the materialsymbols-python repository

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

from kivymd.app import MDApp

KV = '''
```

(continues on next page)

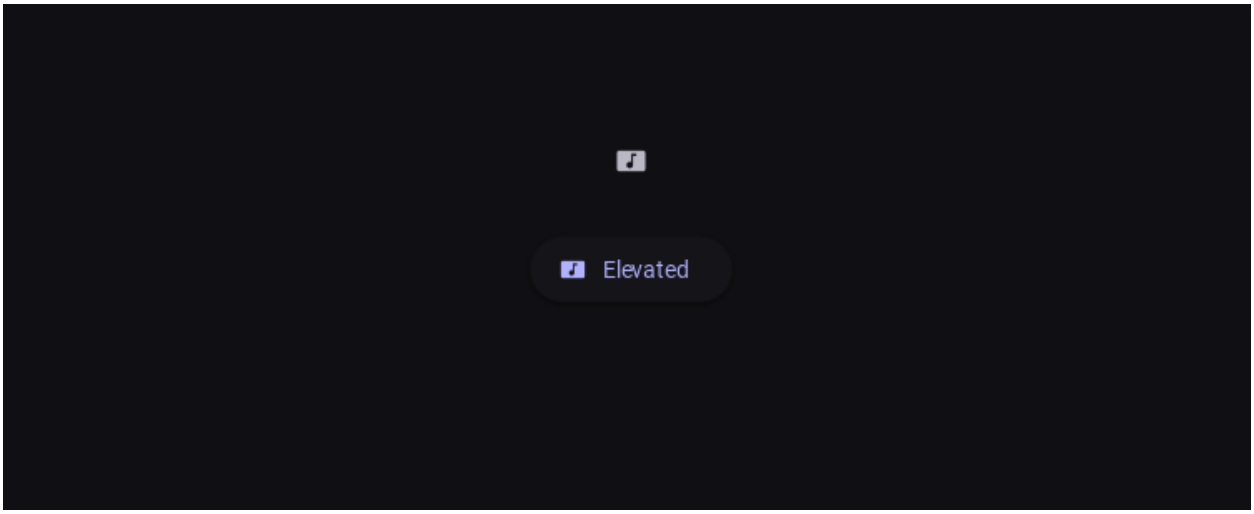
(continued from previous page)

```
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    MDIcon:
        icon: "music_video"
        theme_font_name: "Custom"
        font_name: "MaterialSymbols"
        pos_hint: {"center_x": .5, "center_y": .58}

    MDButton:
        pos_hint: {"center_x": .5, "center_y": .47}

        MDButtonIcon:
            icon: "music_video"
            theme_font_name: "Custom"
            font_name: "MaterialSymbols"

        MDButtonText:
            text: "Elevated"
'''


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

        LabelBase.register(
            name="MaterialSymbols",
            fn_regular="Material_Symbols_Outlined-20-200-1_200.ttf",
        )

        self.theme_cls.font_styles["MaterialSymbols"] = {
            "large": {
                "line-height": 1.64,
                "font-name": "MaterialSymbols",
                "font-size": sp(57),
            },
            "medium": {
                "line-height": 1.52,
                "font-name": "MaterialSymbols",
                "font-size": sp(45),
            },
            "small": {
                "line-height": 1.44,
                "font-name": "MaterialSymbols",
                "font-size": sp(36),
            },
        }

        return Builder.load_string(KV)
```

(continues on next page)

(continued from previous page)

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



API - kivymd.uix.label.label

class kivymd.uix.label.label.**MDLabel**(**args*, ***kwargs*)

Label class.

For more information, see in the *DeclarativeBehavior* and *ThemableBehavior* and *BackgroundColorBehavior* and Label and *MDAdaptiveWidget* and *TouchBehavior* and *StateLayerBehavior* classes documentation.

Events

on_ref_press
Fired when the user clicks on a word referenced with a [ref] tag in a text markup.

on_copy
Fired when double-tapping on the label.

on_selection
Fired when double-tapping on the label.

on_cancel_selection
Fired when the highlighting is removed from the label text.

font_style

Label font style.

Changed in version 2.0.0.

Available vanilla font_style are: *'Display'*, *'Headline'*, *'Title'*, *'Label'*, *'Body'* `.

font_style is an StringProperty and defaults to *'Body'*.

role

Role of font style.

New in version 2.0.0.

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

role is an StringProperty and defaults to *'large'*.

text

Text of the label.

text is an StringProperty and defaults to ‘’.

text_color

Label text color in (r, g, b, a) or string format.

text_color is an ColorProperty and defaults to *None*.

allow_copy

Allows you to copy text to the clipboard by double-clicking on the label.

New in version 1.2.0.

allow_copy is an BooleanProperty and defaults to *False*.

allow_selection

Allows to highlight text by double-clicking on the label.

New in version 1.2.0.

allow_selection is an BooleanProperty and defaults to *False*.

color_selection

The color in (r, g, b, a) or string format of the text selection when the value of the *allow_selection* attribute is True.

New in version 1.2.0.

color_selection is an ColorProperty and defaults to *None*.

color_deselection

The color in (r, g, b, a) or string format of the text deselection when the value of the *allow_selection* attribute is True.

New in version 1.2.0.

color_deselection is an ColorProperty and defaults to *None*.

is_selected

Is the label text highlighted.

New in version 1.2.0.

is_selected is an BooleanProperty and defaults to *False*.

radius

Label radius.

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

do_selection() → None

cancel_selection() → None

on_double_tap(*touch*, **args*) → None

Fired by double-clicking on the widget.

on_window_touch(**args*) → None

Fired at the on_touch_down event.

on_copy(**args*) → None

Fired when double-tapping on the label.

New in version 1.2.0.

on_selection(**args*) → None

Fired when double-tapping on the label.

New in version 1.2.0.

on_cancel_selection(**args*) → None

Fired when the highlighting is removed from the label text.

New in version 1.2.0.

on_allow_selection(*instance_label*, *selection: bool*) → None

Fired when the *allow_selection* value changes.

on_text_color(*instance_label*, *color: list | str*) → None

Fired when the *text_color* value changes.

on_md_bg_color(*instance_label*, *color: list | str*) → None

Fired when the `md_bg_color` value changes.

on_size(*instance_label*, *size: list*) → None

Fired when the parent window of the application is resized.

update_canvas_bg_pos(*instance_label*, *pos: list*) → None

class kivymd.uix.label.label.**MDIcon**(**args*, ***kwargs*)

Icon class.

For more information, see in the *MDLabel* class documentation.

icon

Label icon name.

icon is an `StringProperty` and defaults to *'blank'*.

source

Path to icon.

source is an `StringProperty` and defaults to *None*.

icon_color

Icon color in (r, g, b, a) or string format.

New in version 2.0.0.

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

icon_color_disabled

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

New in version 2.0.0.

icon_color_disabled is a `ColorProperty` and defaults to *None*.

add_widget(*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.

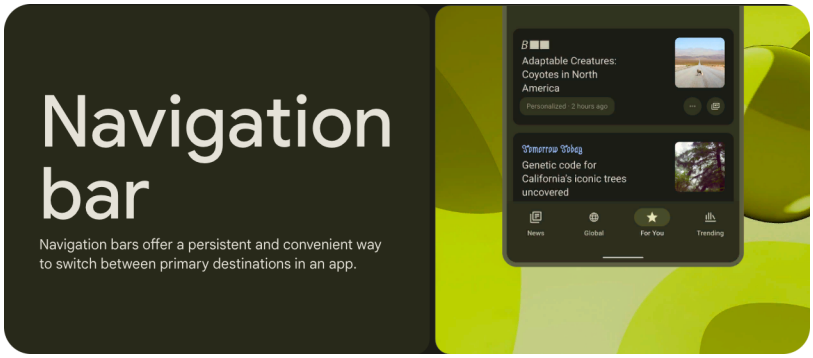
```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

2.3.49 Navigation bar

See also:

Material Design 3 spec, Navigation bar

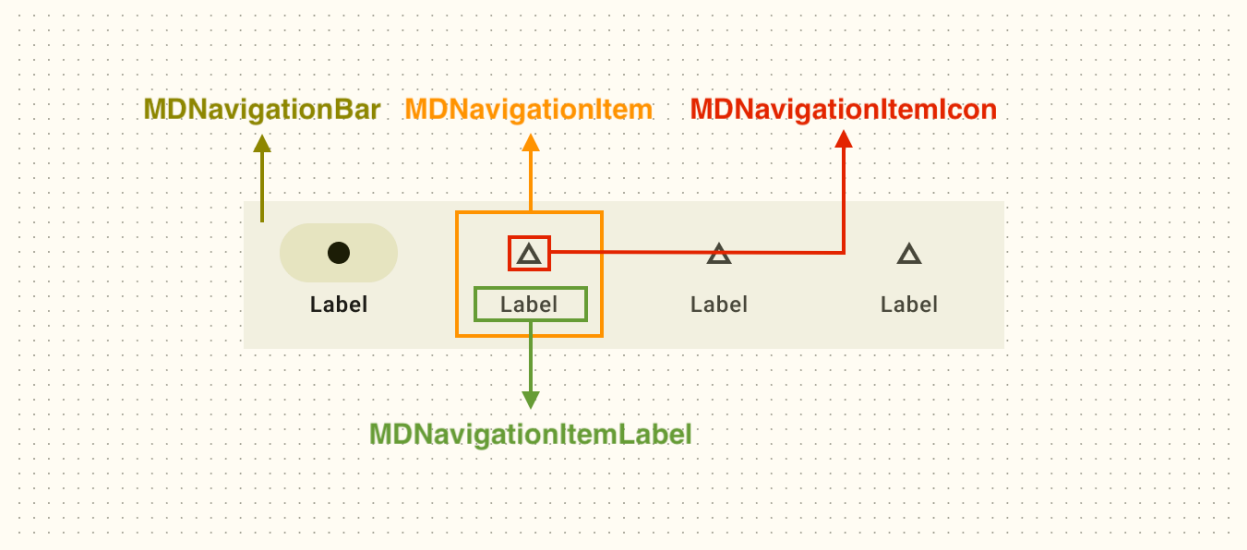
Bottom navigation bars allow movement between primary destinations in an app:



Usage

```
<Root>

    MDNavigationBar:

        MDNavigationItem:

            MDNavigationItemIcon:

            MDNavigationItemLabel:

        [...]
```

Anatomy



Example

Declarative KV style

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

from kivymd.app import MDApp
from kivymd.uix.navigationbar import MDNavigationBar, MDNavigationItem
from kivymd.uix.screen import MDScreen


class BaseMDNavigationItem(MDNavigationItem):
    icon = StringProperty()
    text = StringProperty()
```

(continues on next page)

(continued from previous page)

```
class BaseScreen(MDScreen):
    image_size = StringProperty()


KV = '''
<BaseMDNavigationItem>

    MDNavigationItemIcon:
        icon: root.icon

    MDNavigationItemLabel:
        text: root.text


<BaseScreen>

    FitImage:
        source: f"https://picsum.photos/{root.image_size}/{root.image_size}"
        size_hint: .9, .9
        pos_hint: {"center_x": .5, "center_y": .5}
        radius: dp(24)


MDBoxLayout:
    orientation: "vertical"
    md_bg_color: self.theme_cls.backgroundColor

    MDScreenManager:
        id: screen_manager

        BaseScreen:
            name: "Screen 1"
            image_size: "1024"

        BaseScreen:
            name: "Screen 2"
            image_size: "800"

        BaseScreen:
            name: "Screen 3"
            image_size: "600"


    MDNavigationBar:
        on_switch_tabs: app.on_switch_tabs(*args)

        BaseMDNavigationItem
            icon: "gmail"
            text: "Screen 1"
            active: True
```

(continues on next page)

(continued from previous page)

```
        BaseMDNavigationItem
            icon: "twitter"
            text: "Screen 2"

        BaseMDNavigationItem
            icon: "linkedin"
            text: "Screen 3"
'''


class Example(MDApp):
    def on_switch_tabs(
        self,
        bar: MDNavigationBar,
        item: MDNavigationItem,
        item_icon: str,
        item_text: str,
    ):
        self.root.ids.screen_manager.current = item_text

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

Declarative python style

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

from kivymd.uix.fitimage import FitImage
from kivymd.uix.screen import MDScreen
from kivymd.uix.screenmanager import MDScreenManager
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.navigationbar import (
    MDNavigationBar,
    MDNavigationItem,
    MDNavigationItemLabel,
    MDNavigationItemIcon,
)
from kivymd.app import MDApp


class BaseMDNavigationItem(MDNavigationItem):
    icon = StringProperty()
    text = StringProperty()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.add_widget(MDNavigationItemIcon(icon=self.icon))
        self.add_widget(MDNavigationItemLabel(text=self.text))
```

(continues on next page)

(continued from previous page)

```
class BaseScreen(MDScreen):
    image_size = StringProperty()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.add_widget(
            FitImage(
                source=f"https://picsum.photos/{self.image_size}/{self.image_size}",
                size_hint=(0.9, 0.9),
                pos_hint={"center_x": 0.5, "center_y": 0.5},
                radius=dp(24),
            ),
        )


class Example(MDApp):
    def on_switch_tabs(
        self,
        bar: MDNavigationBar,
        item: MDNavigationItem,
        item_icon: str,
        item_text: str,
    ):
        self.root.get_ids().screen_manager.current = item_text

    def build(self):
        return MDBoxLayout(
            MDScreenManager(
                BaseScreen(
                    name="Screen 1",
                    image_size="1024",
                ),
                BaseScreen(
                    name="Screen 2",
                    image_size="800",
                ),
                BaseScreen(
                    name="Screen 3",
                    image_size="600",
                ),
                id="screen_manager",
            ),
            MDNavigationBar(
                BaseMDNavigationItem(
                    icon="gmail",
                    text="Screen 1",
                    active=True,
                ),
                BaseMDNavigationItem(
                    icon="twitter",
```

(continues on next page)

(continued from previous page)

```
                    text="Screen 2",
                ),
                BaseMDNavigationItem(
                    icon="linkedin",
                    text="Screen 3",
                ),
                on_switch_tabs=self.on_switch_tabs,
            ),
            orientation="vertical",
            md_bg_color=self.theme_cls.backgroundColor,
        )


Example().run()
```

API break

1.2.0 version

```
from kivy.lang import Builder

from kivymd.app import MDApp


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

    MDBottomNavigation:

        MDBottomNavigationItem:
            name: 'screen 1'
            text: 'Mail'
            icon: 'gmail'
            badge_icon: "numeric-10"

            MDLabel:
                text: 'Screen 1'
                halign: 'center'

        MDBottomNavigationItem:
            name: 'screen 2'
            text: 'Twitter'
            icon: 'twitter'

            MDLabel:
                text: 'Screen 2'
```

(continues on next page)

(continued from previous page)

```
                halign: 'center'
'''
        )


Example().run()
```

2.0.0 version

MDNavigationBar in version 2.0.0 no longer provides a screen manager for content placement. You have to implement it yourself. This is due to the fact that when using MDNavigationBar and MDTabs widgets at the same time, there were conflicts between their screen managers.

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

from kivymd.app import MDApp
from kivymd.uix.navigationbar import MDNavigationBar, MDNavigationItem
from kivymd.uix.screen import MDScreen


class BaseMDNavigationItem(MDNavigationItem):
    icon = StringProperty()
    text = StringProperty()


class BaseScreen(MDScreen):
    ...


KV = '''
<BaseMDNavigationItem>

    MDNavigationItemIcon:
        icon: root.icon

    MDNavigationItemLabel:
        text: root.text


<BaseScreen>

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


MDBoxLayout:
    orientation: "vertical"
    md_bg_color: self.theme_cls.backgroundColor
```

(continues on next page)

(continued from previous page)

```
    MDScreenManager:
        id: screen_manager

        BaseScreen:
            name: "Screen 1"

        BaseScreen:
            name: "Screen 2"


    MDNavigationBar:
        on_switch_tabs: app.on_switch_tabs(*args)

        BaseMDNavigationItem
            icon: "gmail"
            text: "Screen 1"
            active: True

        BaseMDNavigationItem
            icon: "twitter"
            text: "Screen 2"
'''


class Example(MDApp):
    def on_switch_tabs(
        self,
        bar: MDNavigationBar,
        item: MDNavigationItem,
        item_icon: str,
        item_text: str,
    ):
        self.root.ids.screen_manager.current = item_text

    def build(self):
        return Builder.load_string(KV)


Example().run()
```

API - kivymd.uix.navigationbar.navigationbar

class kivymd.uix.navigationbar.navigationbar.**MDNavigationItemLabel**(**args*, ***kwargs*)

Implements a text label for the *MDNavigationItem* class.

New in version 2.0.0.

For more information, see in the *MDLabel* class documentation.

text_color_active

Item icon color in (r, g, b, a) or string format.

text_color_active is a ColorProperty and defaults to *None*.

text_color_normal

Item icon color in (r, g, b, a) or string format.

text_color_normal is a ColorProperty and defaults to *None*.

class kivymd.uix.navigationbar.navigationbar.**MDNavigationItemIcon**(**args*, ***kwargs*)

Implements a icon for the *MDNavigationItem* class.

New in version 2.0.0.

For more information, see in the *MDIcon* class documentation.

icon_color_active

Item icon color in (r, g, b, a) or string format.

icon_color_active is a ColorProperty and defaults to *None*.

icon_color_normal

Item icon color in (r, g, b, a) or string format.

icon_color_normal is a ColorProperty and defaults to *None*.

class kivymd.uix.navigationbar.navigationbar.**MDNavigationItem**(**args*, ***kwargs*)

Bottom item class.

For more information, see in the *DeclarativeBehavior* and *RectangularRippleBehavior* and AnchorLayout and ButtonBehavior classes documentation.

Changed in version 2.0.0: Rename class from *MDBottomNavigationItem* to *MDNavigationItem*.

active

Indicates if the bar item is active or inactive.

active is a BooleanProperty and defaults to *False*.

indicator_color

The background color in (r, g, b, a) or string format of the highlighted item.

New in version 1.0.0.

Changed in version 2.0.0: Rename property from *selected_color_background* to *indicator_color*.

indicator_color is an ColorProperty and defaults to *None*.

indicator_transition

Animation type of the active element indicator.

indicator_transition is an StringProperty and defaults to *'in_out_sine'*.

indicator_duration

Duration of animation of the active element indicator.

indicator_duration is an NumericProperty and defaults to *0.1*.

on_active(*instance*, *value*) → None

Fired when the values of *active* change.

on_release() → None

Fired when clicking on a panel item.

add_widget(*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)
```

class kivymd.uix.navigationbar.navigationbar.**MDNavigationBar**(**args*, ***kwargs*)

A navigation bar class.

For more information, see in the *CommonElevationBehavior* and *MDBoxLayout* classes documentation.

Events

on_switch_tabs
Fired when switching tabs.

New in version 1.0.0.

Changed in version 2.0.0: Rename class from *MDBottomNavigation* to *MDNavigationBar*.

set_bars_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.

set_bars_color is an BooleanProperty and defaults to *False*.

set_active_item(*item:* MDNavigationItem) → None

Sets the currently active element on the panel.

set_status_bar_color(*interval: int | float*) → None

Sets the color of the lower system navigation bar.

on_switch_tabs(*item:* MDNavigationItem, *item_icon: str*, *item_text: str*) → None

Fired when switching tabs.

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(*instance*, *size: list*) → None

Called when the application screen size changes.

get_real_device_type() → str

Returns the device type - 'mobile', 'tablet' or 'desktop'.

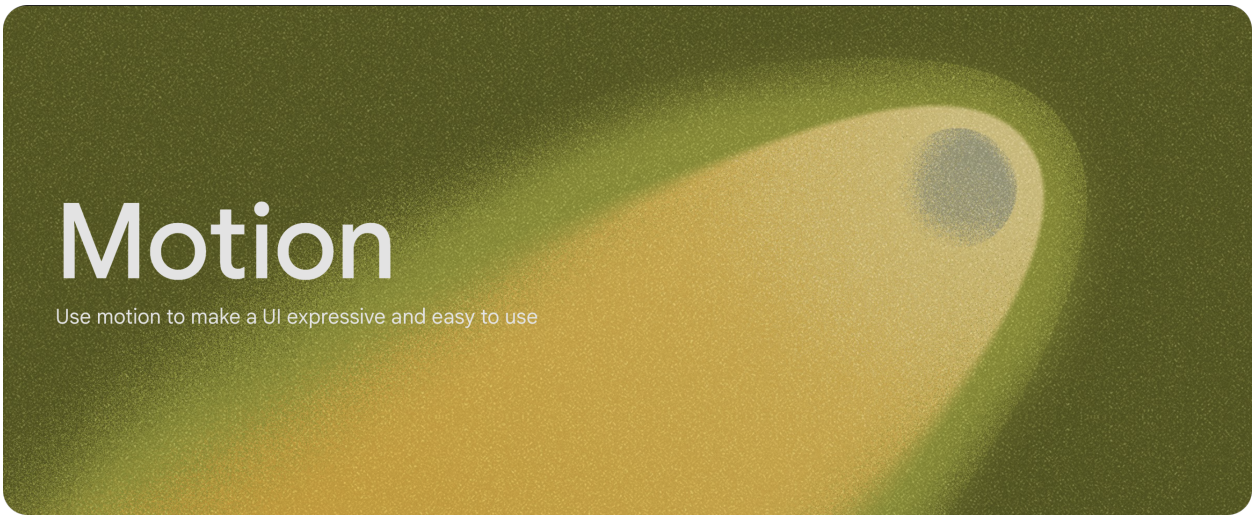
get_window_width_resizing_direction() → str

Return window width resizing direction - 'left' or 'right'.

2.5 Behaviors

2.5.1 Motion

Use motion to make a UI expressive and easy to use.



New in version 1.2.0.

Classes of the *Motion* type implement the display behavior of widgets such as dialogs, dropdown menu, snack bars, and so on.

API - kivymd.uix.behaviors.motion_behavior

class kivymd.uix.behaviors.motion_behavior.**MotionBase**

Base class for widget display movement behavior.

show_transition

The type of transition of the widget opening.

show_transition is a StringProperty and defaults to *'linear'*.

show_duration

Duration of widget display transition.

show_duration is a NumericProperty and defaults to *0.2*.

hide_transition

The type of transition of the widget closing.

hide_transition is a StringProperty and defaults to *'linear'*.

hide_duration

Duration of widget closing transition.

hide_duration is a NumericProperty and defaults to *0.2*.

class kivymd.uix.behaviors.motion_behavior.**MotionDropDownMenuBehavior**(***kwargs*)

Base class for the dropdown menu movement behavior.

For more information, see in the *MotionBase* class documentation.

show_transition

The type of transition of the widget opening.

show_transition is a StringProperty and defaults to *'out_back'*.

show_duration

Duration of widget display transition.

show_duration is a NumericProperty and defaults to *0.2*.

hide_transition

The type of transition of the widget closing.

hide_transition is a StringProperty and defaults to *'out_cubic'*.

set_opacity() → None

set_scale() → None

on_dismiss() → None

on_open(**args*)

on__opacity(*instance*, *value*)

on__scale_x(*instance*, *value*)

on__scale_y(*instance*, *value*)

class kivymd.uix.behaviors.motion_behavior.**MotionExtendedFabButtonBehavior**

Base class for extended Fab button movement behavior.

For more information, see in the *MotionBase* class documentation.

show_transition

The type of transition of the widget opening.

show_transition is a StringProperty and defaults to *'out_circ'*.

shift_transition

Text label transition.

shift_transition is a StringProperty and defaults to *'out_sine'*.

show_duration

Duration of widget display transition.

show_duration is a NumericProperty and defaults to *0.3*.

hide_transition

The type of transition of the widget closing.

hide_transition is a StringProperty and defaults to *'linear'*.

hide_duration

Duration of widget closing transition.

hide_duration is a NumericProperty and defaults to *0.2*.

collapse(**args*) → None

Collapses the button.

expand(**args*) → None

Expands the button.

set_opacity_text_button(*value: int*) → None

class kivymd.uix.behaviors.motion_behavior.**MotionDialogBehavior**

Base class for dialog movement behavior.

For more information, see in the *ScaleBehavior* *MotionBase* classes documentation.

show_transition

The type of transition of the widget opening.

show_transition is a StringProperty and defaults to *'out_expo'*.

show_button_container_transition

The type of transition of the widget opening.

show_button_container_transition is a StringProperty and defaults to *'out_circ'*.

hide_transition

The type of transition of the widget opening.

show_transition is a StringProperty and defaults to *'hide_transition'*.

show_duration

Duration of widget display transition.

show_duration is a NumericProperty and defaults to *0.2*.

on_dismiss(**args*)

Fired when a dialog closed.

on_open(**args*)

Fired when a dialog opened.

class kivymd.uix.behaviors.motion_behavior.**MotionTimePickerBehavior**

Base class for time picker movement behavior.

For more information, see in the MotionPickerBehavior class documentation.

class kivymd.uix.behaviors.motion_behavior.**MotionDatePickerBehavior**

Base class for date picker movement behavior.

For more information, see in the MotionPickerBehavior class documentation.

class kivymd.uix.behaviors.motion_behavior.**MotionShackBehavior**

The base class for the behavior of the movement of snack bars.

For more information, see in the *MotionBase* class documentation.

on_dismiss(**args*)

Fired when a snackbar closed.

on_open(**args*)

Fired when a snackbar opened.

2.5.2 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() → None

Grow effect animation.

shake() → None

Shake effect animation.

wobble() → None

Wobble effect animation.

twist() → None

Twist effect animation.

shrink() → None

Shrink effect animation.

on_touch_up(**args*)

2.5.3 Rotate

New in version 1.1.0.

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)


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 kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import RotateBehavior
from kivymd.uix.boxlayout import MDBoxLayout

KV = '''
MDScreen:

    RotateBox:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_rotate(self)
        md_bg_color: "red"
'''


class RotateBox(ButtonBehavior, RotateBehavior, MDBoxLayout):
    pass


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

    def change_rotate(self, instance_button: RotateBox) -> None:
        Animation(rotate_value_angle=45, d=0.3).start(instance_button)


Test().run()
```

Warning: Do not use *RotateBehavior* class with classes that inherited` from *CommonElevationBehavior* class. *CommonElevationBehavior* classes by default contains attributes for rotate widget.

API - kivymd.uix.behaviors.rotate_behavior

class kivymd.uix.behaviors.rotate_behavior.**RotateBehavior**

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 ListProperty and defaults to *(0, 0, 1)*.

2.5.4 Scale

New in version 1.1.0.

Base class for controlling the scale of the widget.

Note: See kivy.graphics.Rotate 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
'''


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(
```

(continues on next page)

(continued from previous page)

```
            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 kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import ScaleBehavior
from kivymd.uix.boxlayout import MDBoxLayout

KV = '''
MDScreen:

    ScaleBox:
        size_hint: .5, .5
        pos_hint: {"center_x": .5, "center_y": .5}
        on_release: app.change_scale(self)
        md_bg_color: "red"
'''


class ScaleBox(ButtonBehavior, ScaleBehavior, MDBoxLayout):
    pass


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

    def change_scale(self, instance_button: ScaleBox) -> 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()
```

Warning: Do not use *ScaleBehavior* class with classes that inherited` from *CommonElevationBehavior* class. *CommonElevationBehavior* classes by default contains attributes for scale widget.

API - kivymd.uix.behaviors.scale_behavior

class kivymd.uix.behaviors.scale_behavior.**ScaleBehavior**

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*.

scale_value_center

Origin of the scale.

New in version 1.2.0.

The format of the origin can be either (x, y) or (x, y, z).

scale_value_center is an NumericProperty and defaults to *[]*.

2.5.5 ToggleButton

This behavior must always be inherited after the button's Widget class since it works with the inherited properties of the button class.

example:

```
class MyToggleButtonWidget(MDFlatButton, MDToggleButton):
    # [...]
    pass
```

Declarative KV style

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors.toggle_behavior import MDToggleButton
from kivymd.uix.button import MDFlatButton

KV = '''
MDScreen:
```

(continues on next page)

(continued from previous page)

```
    MDBoxLayout:
        adaptive_size: True
        spacing: "12dp"
        pos_hint: {"center_x": .5, "center_y": .5}

        MyToggleButton:
            text: "Show ads"
            group: "x"

        MyToggleButton:
            text: "Do not show ads"
            group: "x"

        MyToggleButton:
            text: "Does not matter"
            group: "x"
'''


class MyToggleButton(MDFlatButton, MDToggleButton):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.background_down = self.theme_cls.primary_color


class Test(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return Builder.load_string(KV)


Test().run()
```

Declarative python style

```
from kivymd.app import MDApp
from kivymd.uix.behaviors.toggle_behavior import MDToggleButton
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDFlatButton
from kivymd.uix.screen import MDScreen


class MyToggleButton(MDFlatButton, MDToggleButton):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.background_down = self.theme_cls.primary_color


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

(continues on next page)

(continued from previous page)

```
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.primary_palette = "Orange"
        return (
            MDScreen(
                MDBoxLayout(
                    MyToggleButton(
                        text="Show ads",
                        group="x",
                    ),
                    MyToggleButton(
                        text="Do not show ads",
                        group="x",
                    ),
                    MyToggleButton(
                        text="Does not matter",
                        group="x",
                    ),
                    adaptive_size=True,
                    spacing="12dp",
                    pos_hint={"center_x": .5, "center_y": .5},
                ),
            )
        )


Test().run()
```

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.**MDToggleButtonBehavior**(***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]*.

2.5.6 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+HoverBehavior>
```

In *python file*:

```
class HoverItem(MDBoxLayout, 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 RelativeLayout, 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

KV = '''
MDScreen
    md_bg_color: self.theme_cls.backgroundColor

    MDBoxLayout:
        id: box
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: .8, .8
        md_bg_color: self.theme_cls.secondaryContainerColor
'''


class HoverItem(MDBoxLayout, 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 = "white"

    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.secondaryContainerColor


class Example(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


Example().run()
```

API - kivymd.uix.behaviors.hover_behavior

class kivymd.uix.behaviors.hover_behavior.**HoverBehavior**(**args*, ***kwargs*)

Events

on_enter
Fired when mouse enters the bbox of the widget and the widget is visible.

on_leave
Fired 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(**args*)

on_enter()

Fired when mouse enter the bbox of the widget.

on_leave()

Fired when the mouse goes outside the widget border.

2.5.7 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*)

background

Background image path.

background is a StringProperty and defaults to ‘’.

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 *[0, 0, 0, 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

on_md_bg_color(*instance*, *color: list | str*)

Fired when the values of *md_bg_color* change.

update_background_origin(*instance*, *pos: list*) → None

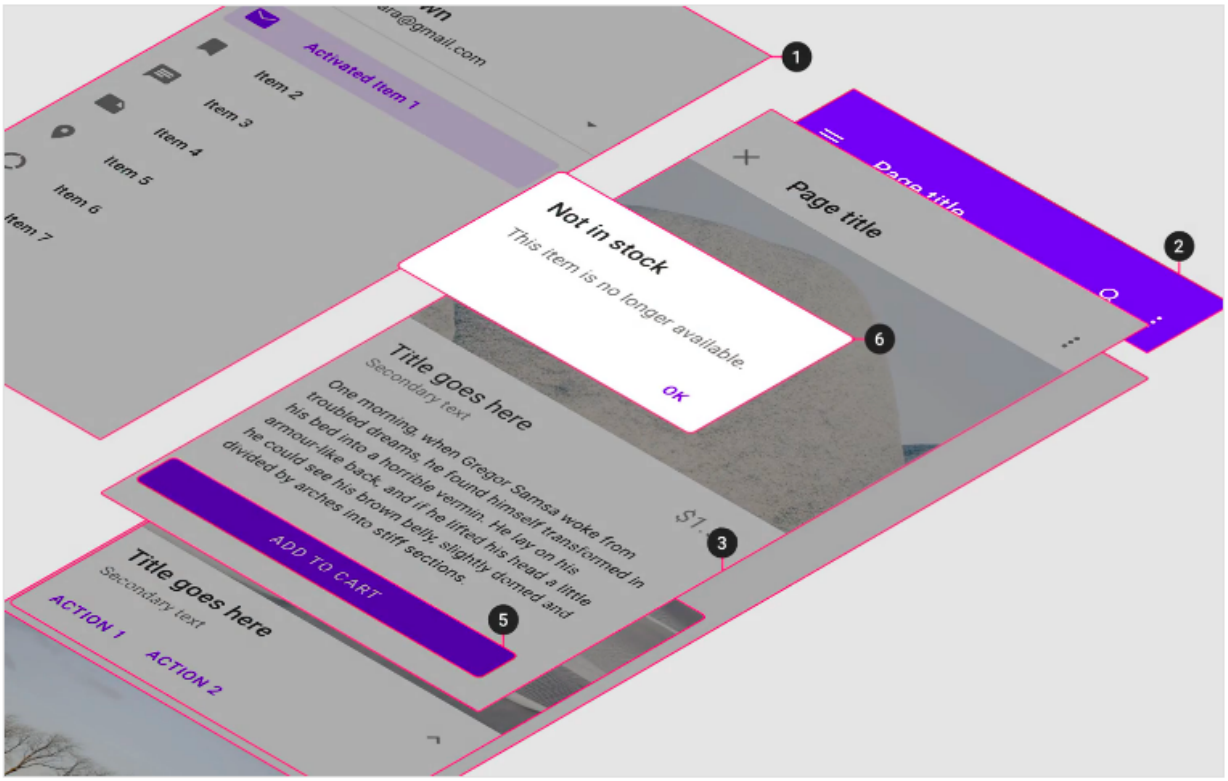
Fired when the values of pos change.

2.5.8 Elevation

See also:

Material Design spec, Elevation

Elevation is the relative distance between two surfaces along the z-axis.



To create an elevation effect, use the *CommonElevationBehavior* class. For example, let's create a button with a rectangular elevation effect:

Declarative style with KV

```
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import (
    RectangularRippleBehavior,
    BackgroundColorBehavior,
    CommonElevationBehavior,
)

KV = '''
<ElevationWidget>
    size_hint: None, None
    size: "250dp", "50dp"


MDScreen:

    # With elevation effect
    ElevationWidget:
        pos_hint: {"center_x": .5, "center_y": .6}
        elevation: 4
        shadow_offset: 0, -6
        shadow_softness: 4

    # Without elevation effect
    ElevationWidget:
        pos_hint: {"center_x": .5, "center_y": .4}
'''


class ElevationWidget(
    RectangularRippleBehavior,
    CommonElevationBehavior,
    ButtonBehavior,
    BackgroundColorBehavior,
):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.md_bg_color = "red"


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


Example().run()
```

Declarative python style

```
from kivy.uix.behaviors import ButtonBehavior
```

(continues on next page)

(continued from previous page)

```
from kivymd.app import MDApp
from kivymd.uix.behaviors import (
    RectangularRippleBehavior,
    BackgroundColorBehavior,
    CommonElevationBehavior,
)
from kivymd.uix.screen import MDScreen


class ElevationWidget(
    RectangularRippleBehavior,
    CommonElevationBehavior,
    ButtonBehavior,
    BackgroundColorBehavior,
):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.md_bg_color = "red"
        self.size_hint = (None, None)
        self.size = ("250dp", "50dp")


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                ElevationWidget(
                    pos_hint={"center_x": .5, "center_y": .6},
                    elevation=4,
                    shadow_softness=4,
                    shadow_offset=(0, -6),
                ),
                ElevationWidget(
                    pos_hint={"center_x": .5, "center_y": .4},
                ),
            )
        )


Example().run()
```

Warning: If before the KivyMD 1.1.0 library version you used the elevation property with an average value of *12* for the shadow, then starting with the KivyMD 1.1.0 library version, the average value of the elevation property will be somewhere *4*.

Similarly, create a circular button:

Declarative style with KV

```
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import CircularRippleBehavior, CommonElevationBehavior
from kivymd.uix.floatlayout import MDFloatLayout

KV = '''
<CircularElevationButton>
    size_hint: None, None
    size: "100dp", "100dp"
    radius: self.size[0] / 2
    shadow_radius: self.radius[0]
    md_bg_color: "red"

    MDIcon:
        icon: "hand-heart"
        halign: "center"
        valign: "center"
        pos_hint: {"center_x": .5, "center_y": .5}
        size: root.size
        pos: root.pos
        font_size: root.size[0] * .6
        theme_text_color: "Custom"
        text_color: "white"


MDScreen:

    CircularElevationButton:
        pos_hint: {"center_x": .5, "center_y": .6}
```

(continues on next page)

(continued from previous page)

```
        elevation: 4
        shadow_softness: 4
'''


class CircularElevationButton(
    CommonElevationBehavior,
    CircularRippleBehavior,
    ButtonBehavior,
    MDFloatLayout,
):
    pass


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


Example().run()
```

Declarative python style

```
from kivy.metrics import dp
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import CircularRippleBehavior, CommonElevationBehavior
from kivymd.uix.floatlayout import MDFloatLayout
from kivymd.uix.label import MDIcon
from kivymd.uix.screen import MDScreen


class CircularElevationButton(
    CommonElevationBehavior,
    CircularRippleBehavior,
    ButtonBehavior,
    MDFloatLayout,
):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.size_hint = (None, None)
        self.size = (dp(100), dp(100))
        self.radius = dp(100) / 2
        self.shadow_radius = dp(100) / 2
        self.md_bg_color = "red"
        self.add_widget(
            MDIcon(
                icon="hand-heart",
                halign="center",
                valign="center",
                pos_hint={"center_x": .5, "center_y": .5},
```

(continues on next page)

(continued from previous page)

```
                size=self.size,
                theme_text_color="Custom",
                text_color="white",
                font_size=self.size[0] * 0.6,
            )
        )


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                CircularElevationButton(
                    pos_hint={"center_x": .5, "center_y": .5},
                    elevation=4,
                    shadow_softness=4,
                )
            )
        )


Example().run()
```

Animating the elevation

Declarative style with KV

```
from kivy.animation import Animation
from kivy.lang import Builder
from kivy.uix.behaviors import ButtonBehavior

from kivymd.app import MDApp
from kivymd.uix.behaviors import CommonElevationBehavior, RectangularRippleBehavior
from kivymd.uix.widget import MDWidget

KV = '''
MDScreen:
```

(continues on next page)

(continued from previous page)

```
    ElevatedWidget:
        pos_hint: {'center_x': .5, 'center_y': .5}
        size_hint: None, None
        size: 100, 100
        md_bg_color: 0, 0, 1, 1
        elevation: 2
        radius: dp(18)
'''


class ElevatedWidget(
    CommonElevationBehavior,
    RectangularRippleBehavior,
    ButtonBehavior,
    MDWidget,
):
    _elev = 0  # previous elevation value

    def on_press(self, *args):
        if not self._elev:
            self._elev = self.elevation
        Animation(elevation=self.elevation + 2, d=0.4).start(self)

    def on_release(self, *args):
        Animation.cancel_all(self, "elevation")
        Animation(elevation=self._elev, d=0.1).start(self)


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


Example().run()
```

Declarative python style

```
from kivy.animation import Animation
from kivy.uix.behaviors import ButtonBehavior
from kivy.metrics import dp

from kivymd.app import MDApp
from kivymd.uix.behaviors import CommonElevationBehavior, RectangularRippleBehavior
from kivymd.uix.screen import MDScreen
from kivymd.uix.widget import MDWidget


class ElevatedWidget(
    CommonElevationBehavior,
    RectangularRippleBehavior,
    ButtonBehavior,
```

(continues on next page)

(continued from previous page)

```
    MDWidget,
):
    _elev = 0  # previous elevation value

    def on_press(self, *args):
        if not self._elev:
            self._elev = self.elevation
        Animation(elevation=self.elevation + 2, d=0.4).start(self)

    def on_release(self, *args):
        Animation.cancel_all(self, "elevation")
        Animation(elevation=self._elev, d=0.1).start(self)


class Example(MDApp):
    def build(self):
        return (
            MDScreen(
                ElevatedWidget(
                    pos_hint={'center_x': .5, 'center_y': .5},
                    size_hint=(None, None),
                    size=(100, 100),
                    md_bg_color="blue",
                    elevation=2,
                    radius=dp(18),
                )
            )
        )


Example().run()
```

API - kivymd.uix.behaviors.elevation

class kivymd.uix.behaviors.elevation.**CommonElevationBehavior**(***kwargs*)

Common base class for rectangular and circular elevation behavior.

For more information, see in the `Widget` class documentation.

elevation_level

Elevation level (values from 0 to 5)

New in version 1.2.0.

elevation_level is an `BoundedNumericProperty` and defaults to *0*.

elevation_levels

Elevation is measured as the distance between components along the z-axis in density-independent pixels (dps).

New in version 1.2.0.

elevation_levels is an DictProperty and defaults to *{0: dp(0), 1: dp(8), 2: dp(23), 3: dp(16), 4: dp(20), 5: dp(24)}*.

elevation

Elevation of the widget.

elevation is an BoundedNumericProperty and defaults to *0*.

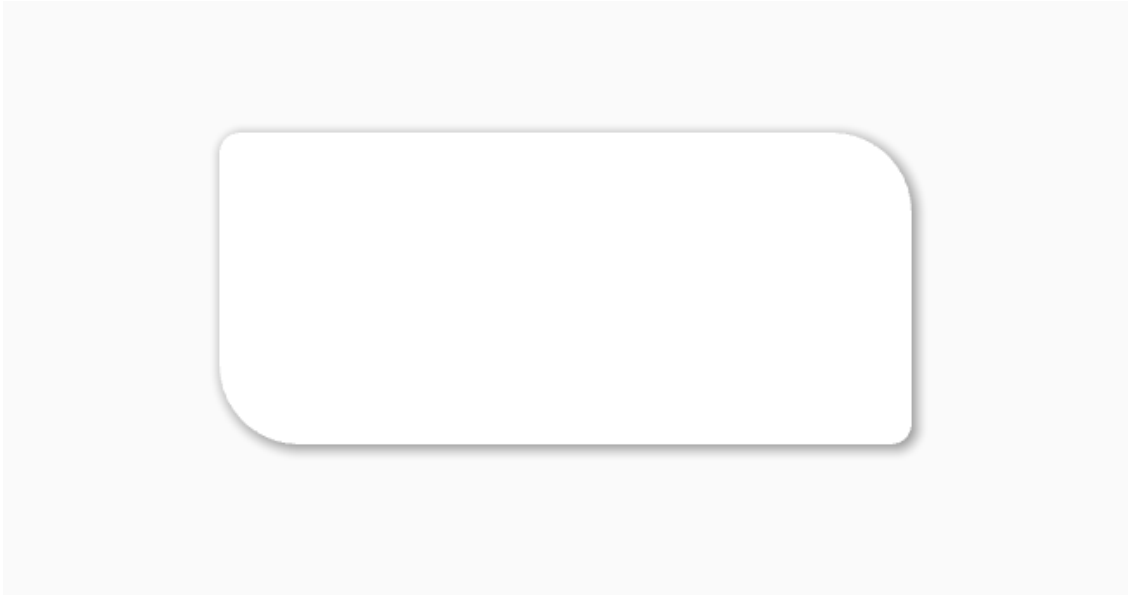
shadow_radius

Radius of the corners of the shadow.

New in version 1.1.0.

You don't have to use this parameter. The radius of the elevation effect is calculated automatically one way or another based on the radius of the parent widget, for example:

```
from kivy.lang import Builder

from kivymd.app import MDApp

KV = '''
MDScreen:

    MDCard:
        radius: dp(12), dp(46), dp(12), dp(46)
        size_hint: .5, .3
        pos_hint: {"center_x": .5, "center_y": .5}
        elevation: 2
        shadow_softness: 4
        shadow_offset: (2, -2)
'''


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


Test().run()
```



shadow_radius is an VariableListProperty and defaults to *[0, 0, 0, 0]*.

shadow_softness

Softness of the shadow.

New in version 1.1.0.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import BackgroundColorBehavior,␣
↪CommonElevationBehavior

KV = '''
<ElevationWidget>
    size_hint: None, None
    size: "250dp", "50dp"


MDScreen:

    ElevationWidget:
        pos_hint: {"center_x": .5, "center_y": .6}
        elevation: 6
        shadow_softness: 6

    ElevationWidget:
        pos_hint: {"center_x": .5, "center_y": .4}
        elevation: 6
        shadow_softness: 12
'''


class ElevationWidget(CommonElevationBehavior, BackgroundColorBehavior):
    def __init__(self, **kwargs):
```

(continues on next page)

(continued from previous page)

```
        super().__init__(**kwargs)
        self.md_bg_color = "blue"


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


Example().run()
```

shadow_softness is an NumericProperty and defaults to *0.0*.

shadow_offset

Offset of the shadow.

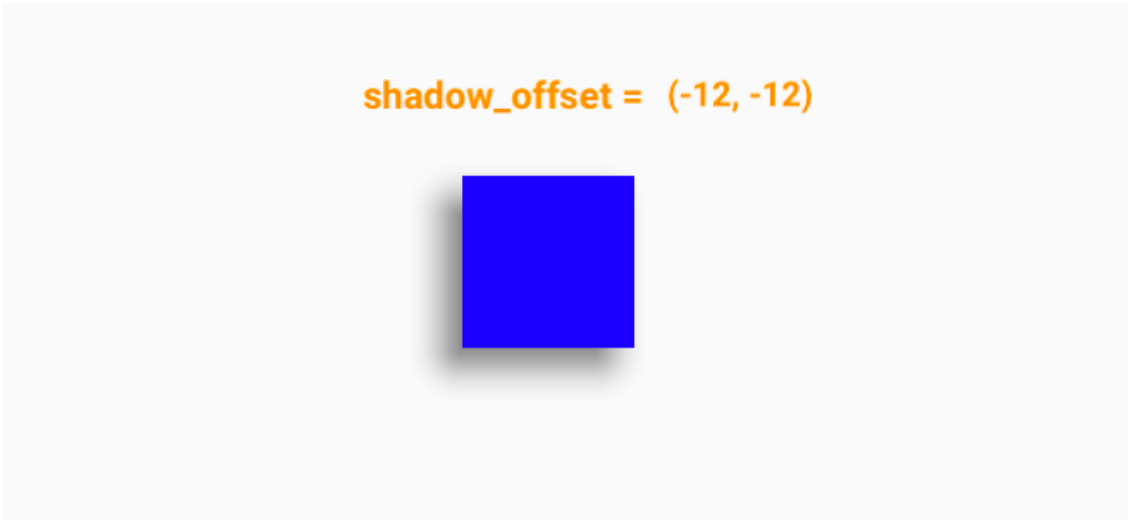
New in version 1.1.0.

```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.uix.behaviors import BackgroundColorBehavior,␣
↪CommonElevationBehavior

KV = '''
<ElevationWidget>
    size_hint: None, None
    size: "100dp", "100dp"


MDScreen:

    ElevationWidget:
        pos_hint: {"center_x": .5, "center_y": .5}
        elevation: 6
        shadow_radius: dp(6)
```

(continues on next page)

(continued from previous page)

```
        shadow_softness: 12
        shadow_offset: -12, -12
'''


class ElevationWidget(CommonElevationBehavior, BackgroundColorBehavior):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.md_bg_color = "blue"


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


Example().run()
```




```
ElevationWidget:
    shadow_offset: 12, -12
```


```
ElevationWidget:
    shadow_offset: 12, 12
```


```
ElevationWidget:
    shadow_offset: -12, 12
```

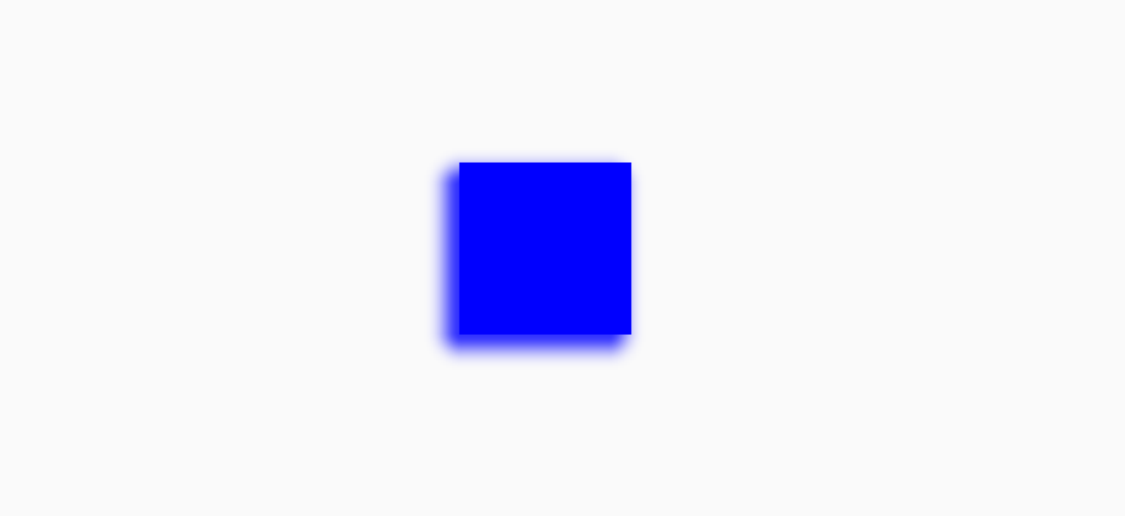
shadow_offset is an ListProperty and defaults to *(0, 0)*.

shadow_color

Offset of the shadow.

New in version 1.1.0.

```
ElevationWidget:
    shadow_color: 0, 0, 1, .8
```



shadow_color is an ColorProperty and defaults to *[0, 0, 0, 0.6]*.

scale_value_x

X-axis value.

New in version 1.2.0.

scale_value_x is an NumericProperty and defaults to *1*.

scale_value_y

Y-axis value.

New in version 1.2.0.

scale_value_y is an NumericProperty and defaults to *1*.

scale_value_z

Z-axis value.

New in version 1.2.0.

scale_value_z is an NumericProperty and defaults to *1*.

scale_value_center

Origin of the scale.

New in version 1.2.0.

The format of the origin can be either (x, y) or (x, y, z).

scale_value_center is an NumericProperty and defaults to *[]*.

rotate_value_angle

Property for getting/setting the angle of the rotation.

New in version 1.2.0.

rotate_value_angle is an NumericProperty and defaults to *0*.

rotate_value_axis

Property for getting/setting the axis of the rotation.

New in version 1.2.0.

rotate_value_axis is an ListProperty and defaults to *(0, 0, 1)*.

2.5.9 State Layer

See also:

Material Design spec, State layers

API - kivymd.uix.behaviors.state_layer_behavior

class kivymd.uix.behaviors.state_layer_behavior.**StateLayerBehavior**(**args*, ***kwargs*)

Focus behavior class.

For more information, see in the HoverBehavior and ButtonBehavior classes documentation.

Events

on_enter
Fired when mouse enters the bbox of the widget AND the widget is visible.

on_leave
Fired when the mouse exits the widget AND the widget is visible.

state_layer_color

The color of the layer state.

state_layer_color is an ColorProperty and defaults to *[0, 0, 0, 0]*.

state_hover

The transparency level of the layer as a percentage when hovering.

state_hover is an NumericProperty and defaults to *0.08*.

state_press

The transparency level of the layer as a percentage when pressed.

state_press is an NumericProperty and defaults to *0.12*.

state_drag

The transparency level of the layer as a percentage when dragged.

state_drag is an NumericProperty and defaults to *0.16*.

icon_button_filled_opacity_value_disabled_container

icon_button_filled_opacity_value_disabled_icon

icon_button_tonal_opacity_value_disabled_container

icon_button_tonal_opacity_value_disabled_icon

icon_button_outlined_opacity_value_disabled_container

icon_button_outlined_opacity_value_disabled_line

icon_button_outlined_opacity_value_disabled_icon

icon_button_standard_opacity_value_disabled_icon

fab_button_opacity_value_disabled_container

fab_button_opacity_value_disabled_icon

button_filled_opacity_value_disabled_container

button_filled_opacity_value_disabled_icon

button_filled_opacity_value_disabled_text

button_tonal_opacity_value_disabled_container

button_tonal_opacity_value_disabled_icon

button_tonal_opacity_value_disabled_text

button_outlined_opacity_value_disabled_container

button_outlined_opacity_value_disabled_line

button_outlined_opacity_value_disabled_icon

button_outlined_opacity_value_disabled_text

button_elevated_opacity_value_disabled_container

button_elevated_opacity_value_disabled_icon

button_elevated_opacity_value_disabled_text

button_text_opacity_value_disabled_icon

button_text_opacity_value_disabled_text

label_opacity_value_disabled_text

card_filled_opacity_value_disabled_state_container

card_outlined_opacity_value_disabled_state_container

card_opacity_value_disabled_state_elevated_container

segmented_button_opacity_value_disabled_container

segmented_button_opacity_value_disabled_container_active

segmented_button_opacity_value_disabled_line

segmented_button_opacity_value_disabled_icon

segmented_button_opacity_value_disabled_text

chip_opacity_value_disabled_container

chip_opacity_value_disabled_text

chip_opacity_value_disabled_icon

switch_opacity_value_disabled_line

switch_opacity_value_disabled_container

switch_thumb_opacity_value_disabled_container

switch_opacity_value_disabled_icon

checkbox_opacity_value_disabled_container

list_opacity_value_disabled_container

list_opacity_value_disabled_leading_avatar

text_field_filled_opacity_value_disabled_state_container

text_field_outlined_opacity_value_disabled_state_container

text_field_opacity_value_disabled_max_length_label

text_field_opacity_value_disabled_helper_text_label

text_field_opacity_value_disabled_hint_text_label

text_field_opacity_value_disabled_leading_icon

text_field_opacity_value_disabled_trailing_icon

text_field_opacity_value_disabled_line

set_properties_widget() → None

Fired *on_release/on_press/on_enter/on_leave* events.

on_disabled(*instance*, *value*) → None

Fired when the *disabled* value changes.

on_enter() → None

Fired when mouse enter the bbox of the widget.

on_leave() → None

Fired when the mouse goes outside the widget border.

2.5.10 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.navigationbar import (
    MDNavigationBar,
    MDNavigationItem,
    MDNavigationItemIcon,
    MDNavigationItemLabel,
)
from kivymd.uix.screen import MDScreen


class Example(MDApp):
    def build(self):
        screen = MDScreen()
        bottom_navigation = MDNavigationBar()

        datas = [
            {"text": "Mail", "icon": "gmail"},
            {"text": "GitHub", "icon": "git"},
            {"text": "LinkedIN", "icon": "linkedin"},
        ]
        for data in datas:
            text = data["text"]
            navigation_item = MDNavigationItem(
                MDNavigationItemIcon(
                    icon=data["icon"],
                ),
```

(continues on next page)

(continued from previous page)

```
                MDNavigationItemLabel(
                    text=text,
                ),
            )
            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 Example(MDApp):
    def build(self):
        return Builder.load_string(
            '''
MDScreen:

    MDNavigationBar:

        MDNavigationItem:

            MDNavigationItemIcon:
                icon: "gmail"

            MDNavigationItemLabel:
                text: "Mail"

        MDNavigationItem:

            MDNavigationItemIcon:
                icon: "git"

            MDNavigationItemLabel:
                text: "GitHub"

        MDNavigationItem:
```

(continues on next page)

(continued from previous page)

```
            MDNavigationItemIcon:
                icon: "linkedin"

            MDNavigationItemLabel:
                text: "LinkedIN"
'''
        )


Example().run()
```

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.navigationbar import (
    MDNavigationBar,
    MDNavigationItemIcon,
    MDNavigationItem,
    MDNavigationItemLabel,
)


class Example(MDApp):
    def build(self):
        return MDNavigationBar(
            MDNavigationItem(
                MDNavigationItemIcon(
                    icon="gmail",
                ),
                MDNavigationItemLabel(
                    text="Mail",
                ),
            ),
            MDNavigationItem(
                MDNavigationItemIcon(
                    icon="twitter",
                ),
                MDNavigationItemLabel(
                    text="Twitter",
                ),
            ),
            MDNavigationItem(
                MDNavigationItemIcon(
                    icon="linkedin",
```

(continues on next page)

(continued from previous page)

```
                ),
                MDNavigationItemLabel(
                    text="LinkedIN",
                ),
            ),
        )


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 kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDButton, MDButtonText
from kivymd.uix.floatlayout import MDFloatLayout
```

(continues on next page)

(continued from previous page)

```
class Example(MDApp):
    def build(self):
        return (
            MDBoxLayout(
                MDFloatLayout(
                    MDButton(
                        MDButtonText(
                            text="Button 1",
                        ),
                        id="button_1",
                        pos_hint={"center_x": 0.5, "center_y": 0.5},
                    ),
                    id="box_container_1",
                ),
                MDBoxLayout(
                    MDFloatLayout(
                        MDButton(
                            MDButtonText(
                                text="Button 2",
                            ),
                            id="button_2",
                            pos_hint={"center_x": 0.5, "center_y": 0.5},
                        ),
                        id="float_container",
                    ),
                    id="box_container_2",
                )
            )
        )

    def on_start(self):
        # {
        #     'button_1': <kivymd.uix.button.button.MDButton object at 0x11d93c9e0>,
        #     'button_2': <kivymd.uix.button.button.MDButton object at 0x11da128f0>,
        #     'float_container': <kivymd.uix.floatlayout.MDFloatLayout object at␣
↪0x11da228f0>,
        #     'box_container_1': <kivymd.uix.floatlayout.MDFloatLayout object at␣
↪0x11d9fc3c0>,
        #     'box_container_2': <kivymd.uix.boxlayout.MDBoxLayout object at 0x11dbf06d0>,
        # }
        print(self.root.get_ids())


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_behavior

class kivymd.uix.behaviors.declarative_behavior.**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 ‘’.

get_ids() → dict

Returns a dictionary of widget IDs defined in Python code that is written in a declarative style.

2.5.11 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"
        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'*.

ripple_effect

Should I use the ripple effect.

ripple_effect is an BooleanProperty and defaults to *True*.

abstract lay_canvas_instructions() → NoReturn

start_ripple() → None

finish_ripple() → None

fade_out(**args*) → None

anim_complete(**args*) → None

Fired when the "fade_out" animation complete.

on_touch_down(*touch*)

call_ripple_animation_methods(*touch*) → None

on_touch_move(*touch*, **args*)

on_touch_up(*touch*)

class kivymd.uix.behaviors.ripple_behavior.**RectangularRippleBehavior**

Class implements a rectangular ripple effect.

For more information, see in the CommonRipple class documentation.

ripple_scale

See *ripple_scale*.

ripple_scale is an NumericProperty and defaults to *2.75*.

lay_canvas_instructions() → None

Adds graphic instructions to the canvas to implement ripple animation.

class kivymd.uix.behaviors.ripple_behavior.**CircularRippleBehavior**

Class implements a circular ripple effect.

For more information, see in the CommonRipple class documentation.

ripple_scale

See *ripple_scale*.

ripple_scale is an NumericProperty and defaults to *1*.

lay_canvas_instructions() → None

2.5.12 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 kivy.properties import StringProperty

from kivymd.app import MDApp
from kivymd.uix.behaviors import TouchBehavior
from kivymd.uix.button import MDButton

KV = '''
<TouchBehaviorButton>
    style: "elevated"

    MDButtonText:
        text: root.text
```

(continues on next page)

(continued from previous page)

```
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

    TouchBehaviorButton:
        text: "TouchBehavior"
        pos_hint: {"center_x": .5, "center_y": .5}
'''


class TouchBehaviorButton(MDButton, TouchBehavior):
    text = StringProperty()

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


Example().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(*widget*, *touch*, **args*)

delete_clock(*widget*, *touch*, **args*)

Removes a key event from *touch.ud*.

on_long_touch(*touch*, **args*)

Fired when the widget is pressed for a long time.

on_double_tap(*touch*, **args*)

Fired by double-clicking on the widget.

on_triple_tap(*touch*, **args*)

Fired by triple clicking on the widget.

2.5.13 Stencil

New in version 1.1.0.

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:

    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.behaviors import StencilBehavior
from kivymd.uix.fitimage import FitImage

KV = '''
#:import os os
#:import images_path kivymd.images_path


Carousel:

    StencilImage:
        size_hint: .9, .8
        pos_hint: {"center_x": .5, "center_y": .5}
        source: os.path.join(images_path, "logo", "kivymd-icon-512.png")
'''


class StencilImage(FitImage, StencilBehavior):
    pass


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


Test().run()
```

API - kivymd.uix.behaviors.stencil_behavior

class kivymd.uix.behaviors.stencil_behavior.**StencilBehavior**

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]*.

2.5.14 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 CommonElevationBehavior
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.behaviors.focus_behavior import FocusBehavior

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, CommonElevationBehavior, 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**(**args*, ***kwargs*)

Focus behavior class.

For more information, see in the `HoverBehavior` and `ButtonBehavior` classes documentation.

Events

on_enter
Fired when mouse enters the bbox of the widget AND the widget is visible.

on_leave
Fired 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*.

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(*dt*)

Before actually updating my velocity, meddle with self.friction to make it appropriate to where I'm at, currently.

on_value(**args*)

Prevent moving beyond my bounds, and update self.scroll

start(*val*, *t=None*)

Start movement with `self.friction` = `self.base_friction`

update(*val*, *t=None*)

Reduce the impact of whatever change has been made to me, in proportion with my current friction.

stop(*val*, *t=None*)

Work out whether I've been flung.

2.7 Changelog

2.7.1 Unreleased

See on GitHub: branch master | compare 2.0.0/master

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

- Bug fixes and other minor improvements.
- [compare] Add supports *Google's Material Design 3* and the *Material You* concept.
- [commit 25f242e] Add the feature to use icons from custom fonts.
- [pull 1584] Implement bitmap scale down.
- [commit 8ba6af9] Fix [issue 1593].
- [commit b34b07f] Fix elevation properties.
- [commit cb01c01] Fixed an infinite loop when typing text fast in the *MDTextfield* widget.

2.7.2 1.1.1

See on GitHub: tag 1.1.1 | compare 1.0.2/1.1.1

```
pip install kivymd==1.1.1
```

- Bug fixes and other minor improvements.
- Add closing_interval parameter to MDCardSwipe class.
- Add implementation of elevation behavior on shaders.
- Add validator property to MDTextField class: the type of text field for entering Email, time, etc. Automatically sets the type of the text field as *error* if the user input does not match any of the set validation types.
- Add theme_style_switch_animation property to animate the colors of the application when switching the color scheme of the application *('Dark/light')*.
- Add theme_style_switch_animation_duration property to duration of the animation of switching the color scheme of the application *("Dark/ light")*.
- Fix memory leak when dynamically adding and removing *KivyMD* widgets.
- Fix slide transition MDBottomNavigation direction.
- Add a default value for the icon attribute of MDApp class.
- Add new properties to MDFileManager class:

  - icon_selection_button - icon that will be used on the directory selection button;
  - background_color_selection_button - background color of the current directory/path selection button;
  - background_color_toolbar - background color of the file manager toolbar;
  - icon_color - color of the folder icon when the *preview* property is set to False;
- Add binds to MDFloatingActionButtonSpeedDial individual buttons;
- Add functionality for using multiple heroes.
- Add shadow_softness_size attribute (value of the softness of the shadow) to CommonElevationBehavior class.
- Optimize of MDDatePicker widget.

2.7.3 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.

2.7.4 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.

2.7.5 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 MDFileManager 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 MDApp 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 MDDataTable widget;
- Added method for update rows to MDDataTable 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 *MDToolbar* 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_right* 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;

2.7.6 0.104.2

See on GitHub: tag 0.104.2 | compare 0.104.1/0.104.2

```
pip install kivymd==0.104.2
```

- Bug fixes and other minor improvements.
- Add *HotReloadViewer* class
- Added features to *Snackbar* class: use padding, set custom button color, elevation
- Add *MDToggleButton* class
- Change to *Material Design Baseline* dark theme spec
- Fix *ReferenceError: weakly-referenced object no longer exists* when start demo application
- Changed the default value for the *theme_text_color* parameter in the *BaseButton* class (to the value *"Primary"*)
- Fix setting of the *text_color_normal* and *text_color_active* parameters - earlier their values did not affect anything
- Fixed the length of the right edge of the border in relation to the hint text when the *MDTextField* is in the *rectangle* mode
- Add *get_tab_list* method to *MDTabs* class
- Add hover behavior when using *MDDropdownMenu* class
- Added the feature to use the *FitImage* component to download images from the network
- The *elevation* value for *RectangularElevationBehavior* and *CircularElevationBehavior* classes after pressing was always set to *2* - fixed
- Methods that implement the ripple effect have always been called twice - fixed
- The *SmartTile* class now uses the *FitImage* class to display images instead of the *Image* class
- Removed dependency on *PIL* library
- Add *hint_bg_color*, *hint_text_color*, *hint_radius* attributes to *MDSlider* class
- Delete *progressloader.py*
- Delete *context_menu.py*
- Added the feature to control the properties of menu items during creation in *MDDropdownMenu* class
- Added the feature to change the number of buttons after creating the *MDFloatingActionButtonSpeedDial* object
- Added the feature to set the *font_name* property for the *MDTabsLabel* class
- Add *MDCarousel* class
- Delete *kivymd/uix/useranimationcard.py*
- Added usage types for *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 *MDRectangleFlatIconButton* 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.

2.7.7 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 *MDDataTable* class

2.7.8 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 *MDTabsLabel* 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

2.7.9 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.

2.7.10 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.

2.7.11 0.102.0

See on GitHub: tag 0.102.0 | compare 0.101.8/0.102.0

```
pip install kivymd==0.102.0
```

- Moved *kivymd.behaviors* to *kivymd.uix.behaviors*.
- Updated [Iconic font](https://github.com/Templarian/MaterialDesign-Webfont) (v4.5.95).
- Added *blank* icon to *icon_definitions*.
- Bug fixes and other minor improvements.

2.7.12 0.101.8

See on GitHub: tag 0.101.8 | compare 0.101.7/0.101.8

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.8.zip
```

- Added *uix* and *behaviors* folder to *package_data*.

2.7.13 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)).

2.7.14 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*.

2.7.15 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.

2.7.16 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.

2.7.17 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.

2.7.18 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.

2.7.19 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.

2.7.20 0.101.0

See on GitHub: tag 0.101.0 | compare 0.100.2/0.101.0

```
pip install https://github.com/kivymd/KivyMD/archive/0.101.0.zip
```

- Added *MDContextMenu* class.
- Added *MDExpansionPanel* class.
- Removed *MDAccordion* and *MDAccordionListItem*. Use *MDExpansionPanel* instead.
- Added *HoverBehavior* class by [Olivier POYEN](https://gist.github.com/opqopq/15c707dc4cffc2b6455f).
- Added markup support for buttons.
- Added *duration* property to *Toast*.
- Added *TextInput*'s events and properties to *MDTextFieldRound*.
- Added feature to resize text field
- Added color property to *MDSeparator* class

- Added [tool](https://github.com/kivymd/KivyMD/blob/master/kivymd/tools/update_icons.py) for updating [Iconic font](https://github.com/Templarian/MaterialDesign-Webfont).
- Updated [Iconic font](https://github.com/Templarian/MaterialDesign-Webfont) (v4.3.95).
- Added new examples for [Kitchen Sink demo](https://github.com/kivymd/KivyMD/tree/master/demos/kitchen_sink).
- Bug fixes and other minor improvements.

2.7.21 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.

2.7.22 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*.

2.7.23 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*.

2.7.24 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*.

2.7.25 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*.

2.7.26 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.

2.7.27 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.

2.7.28 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).

2.7.29 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).

2.7.30 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.

2.7.31 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).

2.7.32 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.

2.8 About

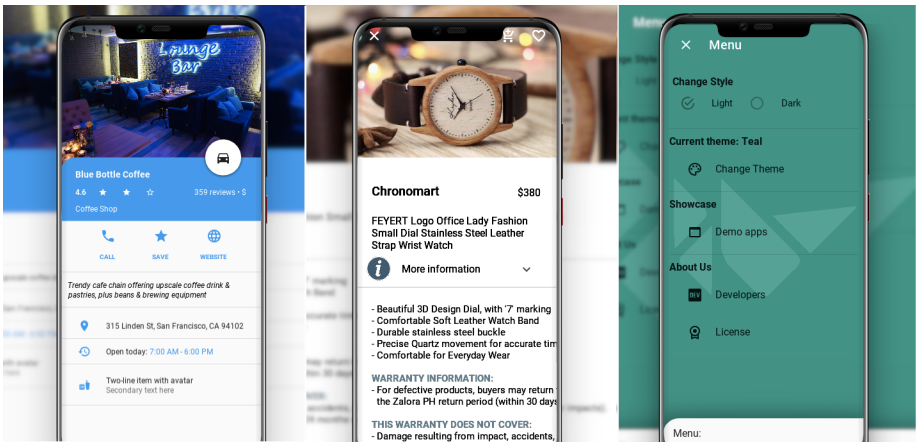
2.8.1 License

Refer to LICENSE.

```
MIT License

Copyright (c) 2024 Andrés Rodríguez and other contributors - KivyMD library up to␣
↪version 0.1.2
Copyright (c) 2024 KivyMD Team and other contributors - KivyMD library version 0.1.3 and␣
↪higher

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.
```

2.9 KivyMD

2.9.1



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.

2.9.2 API - `kivymd`

`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.

2.9.3 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'**

Effects

API - kivymd.effects

Submodules

kivymd.effects.stiffscroll

API - kivymd.effects.stiffscroll

Submodules

kivymd.toast

API - kivymd.toast

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.
# On desktop use `MDSnackbar < https://kivymd.readthedocs.io/en/latest/components/
↪snackbar/>`_
```

(continues on next page)

(continued from previous page)

```
from kivy.lang import Builder

from kivymd.toast import toast
from kivymd.app import MDApp

KV = '''
MDScreen:
    md_bg_color: self.theme_cls.backgroundColor

     MDButton:
         pos_hint:{"center_x": .5, "center_y": .5}
         on_press: app.show_toast()

         MDButtonText:
             text: "Make toast"
'''


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

    def show_toast(self):
        toast("Hello World", True, 80, 200, 0)


Example().run()
```

API - kivymd.toast.androidtoast

kivymd.toast.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;
- **length_long** – 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.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(*args=None*, *namespace=None*)

error(*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()
```

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(*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.

property appname

Return the name of the 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()

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()

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()

Return the root file path.

abstract build_app(*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()

Called when all the application dependencies must be unloaded. Usually happen before a reload

load_app_dependencies()

Load all the application dependencies. This is called before rebuild.

rebuild(**args*, ***kwargs*)

set_error(*exc*, *tb=None*)

bind_key(*key*, *callback*)

Bind a key (keycode) to a callback (cannot be unbind).

enable_autoreload()

Enable autoreload manually. It is activated automatically if "DEBUG" exists in environ. It requires the *watchdog* module.

prepare_foreground_lock()

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(*wid*)

Clear the root container, and set the new approot widget to *wid*.

apply_state(*state*)

Whatever the current state is, reapply the current state.

install_idle(*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(**args*)

Rearm the idle timer.

patch_builder()

on_idle(**args*)

Event fired when the application enter the idle mode.

on_wakeup(**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",
    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 = [(), ()]**

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:

```
kivymd.add_view \
    name_pattern \
    path_to_project \
    name_view
```

Example command:

```
kivymd.add_view \
    MVC \
    /Users/macbookair/Projects \
    NewScreen
```

You can also add new views with responsive behavior to an existing project:

```
kivymd.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.

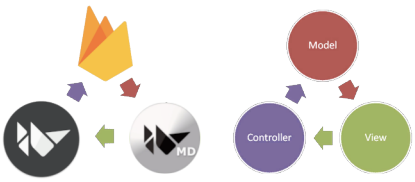
Script creates a project with the MVC pattern

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:

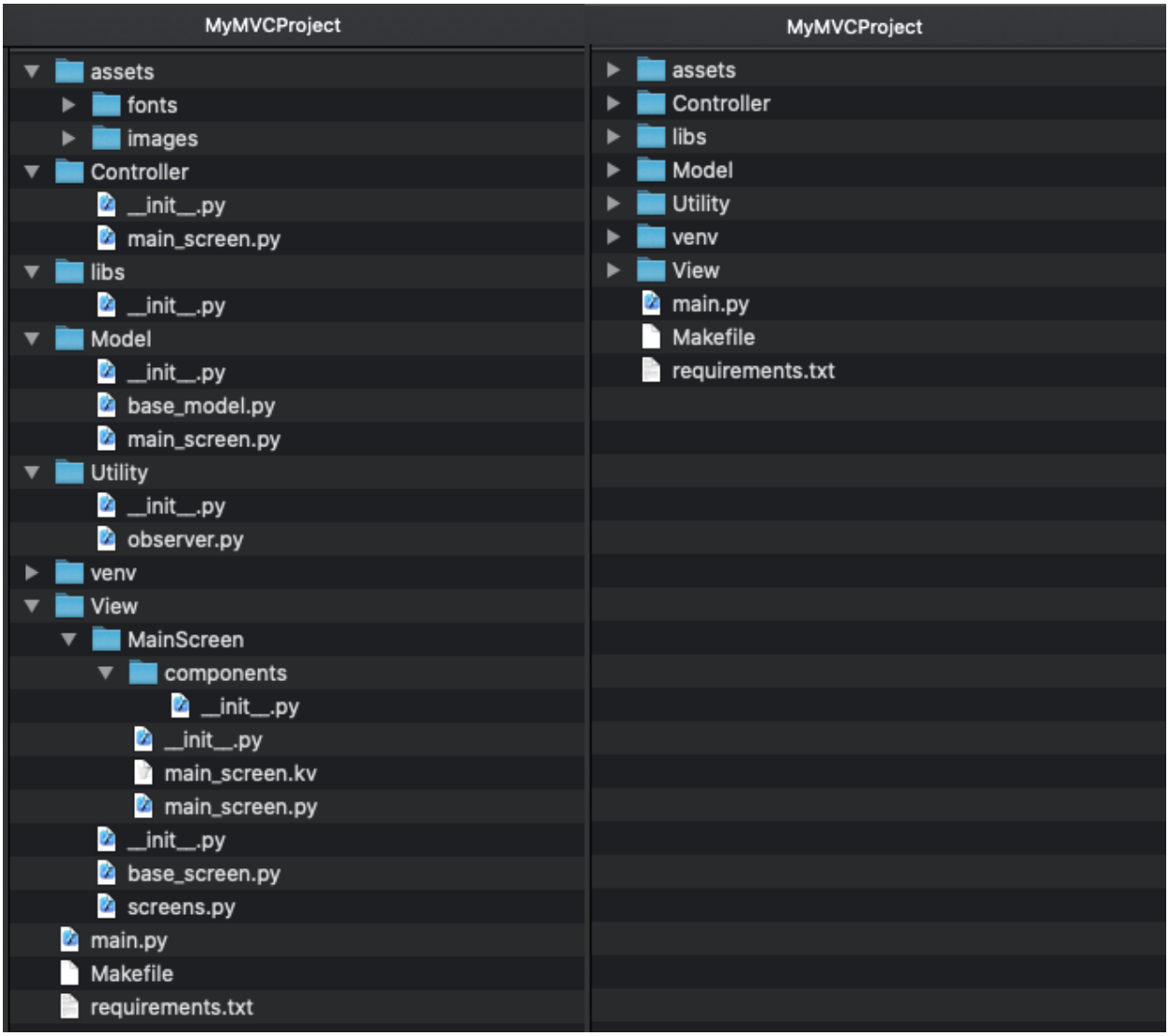
```
kivymd.create_project \
    name_pattern \
    path_to_project \
    name_project \
    python_version \
    kivy_version
```

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'.

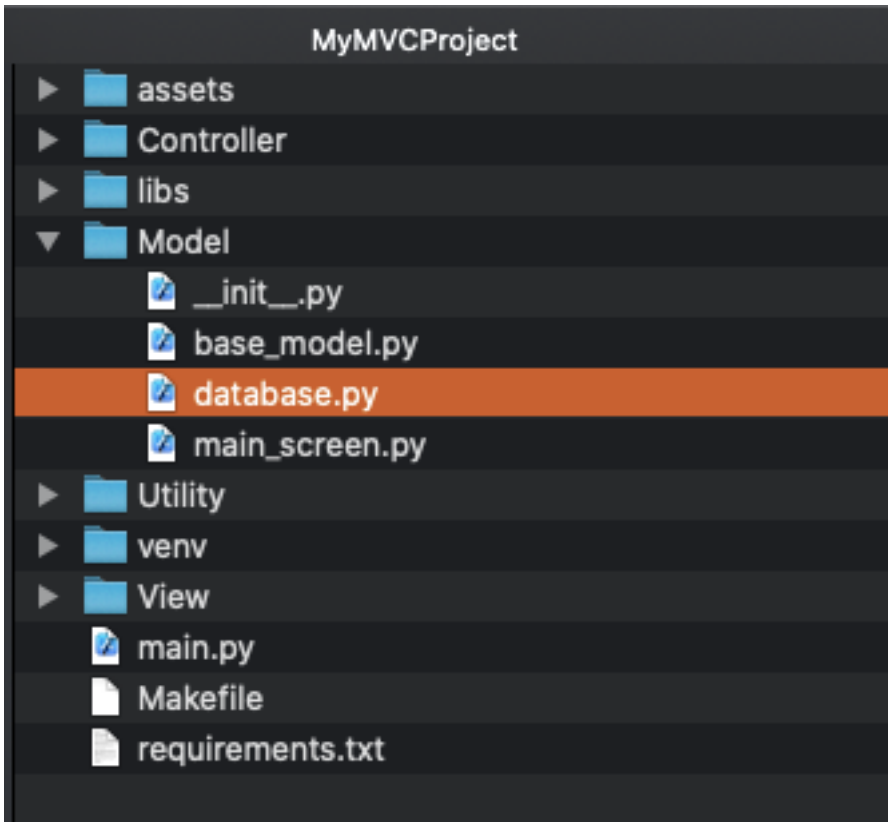
Template command:

```
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 = 'Firebase'

get_data_from_collection(*name_collection: str*) → dict | bool

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(*path_to_file: str*) → dict | bool

Uploads a file to the database. You can upload a file to the database only from a paid account.

get_data_from_collection(*collection_address: str*) → bool | list

Returns data of the selected collection from the database.

delete_doc_from_collection(*collection_address: str*) → bool

Delete data of the selected collection from the database.

Parameters
collection_address – "database_url/id_collection".

add_doc_to_collection(*data: dict*, *collection_address: str*) → bool

Add collection to the database.

edit_data(*collection: dict*, *collection_address: str*, *collection_id: str*) → bool

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(*name*, *func*, *args*, ***kwargs*)

funbind(*name*, *func*, *args*, ***kwargs*)

switch_lang(*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_version_py**(*version*, *is_release*, *test: bool = False*)

Change version in *kivymd/_version.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**

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(*md_widget*, *value: bool*) → None

on_adaptive_width(*md_widget*, *value: bool*) → None

on_adaptive_size(*md_widget*, *value: bool*) → None

Submodules

kivymd.uix.appbar

API - kivymd.uix.appbar

Submodules

kivymd.uix.badge

API - kivymd.uix.badge

Submodules

Behaviors

Modules and classes implementing various behaviors for buttons etc.

API - kivymd.uix.behaviors

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.dialog

API - kivymd.uix.dialog

Submodules

kivymd.uix.divider

API - kivymd.uix.divider

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.navigationbar

API - kivymd.uix.navigationbar

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.datepicker

API - kivymd.uix.pickers.datepicker

Submodules

kivymd.uix.pickers.timepicker

API - kivymd.uix.pickers.timepicker

Submodules

kivymd.uix.progressindicator

API - kivymd.uix.progressindicator

Submodules

kivymd.uix.refreshlayout

API - kivymd.uix.refreshlayout

Submodules

kivymd.uix.segmentedbutton

API - kivymd.uix.segmentedbutton

Submodules

kivymd.uix.selectioncontrol

API - kivymd.uix.selectioncontrol

Submodules

Submodules

kivymd.utils

API - kivymd.utils

Submodules

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*)

Fps monitor class.

For more information, see in the Label class documentation.

updated_interval

FPS refresh rate.

updated_interval is an NumericProperty and defaults to *0.5*.

anchor

Monitor position. Available option are: 'top', 'bottom'.

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

start() → None

Monitor starting.

update_fps(**args*) → None

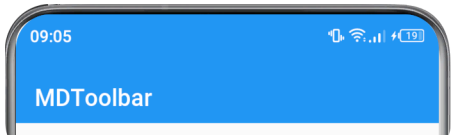
kivymd.utils.set_bars_colors

API - kivymd.utils.set_bars_colors

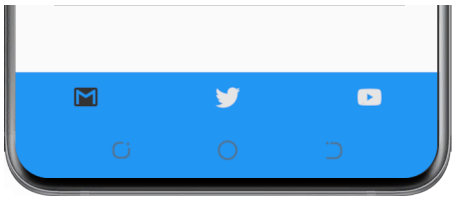
kivymd.utils.set_bars_colors.**set_bars_colors**(*status_bar_color: None | list*, *navigation_bar_color: None | list*, *icons_color: str = 'Light'*)

Sets the color of the status of the StatusBar and NavigationBar.

> **Warning:** Works only on Android devices.

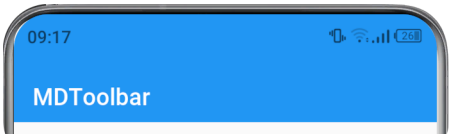



```
from kivy.lang import Builder

from kivymd.app import MDApp
from kivymd.utils.set_bars_colors import set_bars_colors

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

        MDBottomNavigationItem:
            icon: 'gmail'

        MDBottomNavigationItem:
            icon: 'twitter'

        MDBottomNavigationItem:
            icon: 'youtube'
'''


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

    def set_bars_colors(self):
        set_bars_colors(
            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 set_bars_colors(self):
    set_bars_colors(
        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.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y